AD-A034 393 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM, VOLUME I.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-A RADC-TR-76-271-VOL-1 NL

1 OF 3
AD
A034393

Vol 2
A033437

FC.

(12)

RADC-TR-76-271, Vol I (of two)
Final Technical Report
September 1976

MULTICS OLPARS OPERATING SYSTEM

Pattern Analysis and Recognition Corporation

Approved for public release;
distribution unlimited.

ROME AIR DEVELOPMENT CENTER
AIR FORCE SYSTEMS COMMAND
GRIFFISS AIR FORCE BASE, NEW YORK 13441

MULTICS OLPARS OPERATING - VOL I

This report has been reviewed and is approved for publication.

APPROVED: Patricia J. Baskinger

PATRICIA J. BASKINGER
Project Engineer

APPROVED: [illegible]

ROBERT D. KRUTZ, Col, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

Readers of this report are advised that some of the illustrations included herein are relatively poor quality reproductions of computer printouts. They are, however, the best available.

AD-A033 437
PAR-74-25-B

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-76-271, Vol I (of two)	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) MULTICS OLPARS OPERATING SYSTEM, Volume I.		5. TYPE OF REPORT & PERIOD COVERED (9) Final Technical Report, June 1973 — June 1976
		6. PERFORMING ORG. REPORT NUMBER (14) PAR-74-25-A
7. AUTHOR(s) (10) David B. Connell, Richard A. Jackson ~~et al~~ Kermit N. Klingbail		8. CONTRACT OR GRANT NUMBER(s) (15) F30602-75-C-0226, F30602-73-C-0352
9. PERFORMING ORGANIZATION NAME AND ADDRESS Pattern Analysis and Recognition Corporation 228 West Dominick St Rome NY 13440		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55971309
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISCP) Griffiss AFB NY 13441	(12) 219p.	12. REPORT DATE (11) Sep[redacted] 1976
		13. NUMBER OF PAGES 218
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same (16) 5597 (17) 13		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

(18) RADC (19) TR-76-271-Vol-1

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Patricia J. Baskinger (ISCP)

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Pattern Recognition
Pattern Analysis
Decision Theory
Classification
Clustering
Measurement Evaluation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The development of interactive graphics computer systems for use in detection, identification, and transformation of patterns contained in high-dimensional data has been a continuing program at the Rome Air Development Center since 1968 (RADC-TR-70-139; RADC-TR-71-177; RADC-TR-73-241). This long standing effort has resulted in the implementation of OLPARS (the On-Line Pattern Analysis and Recognition System), IFES (the Image Feature Extraction System), and WPS (the Waveform Processing System). This report contains detailed design and

over

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

390 101

user-oriented information related to MOOS (the MULTICS OLPARS Operating System), an advanced version of OLPARS currently resident upon the Honeywell 6180 MULTICS computer system. The currently operational system represents an implemented version of the operations described in a previous report (RADC-TR-73-241); appropriate selections of that report are retained within this document. This report contains brief descriptions of the MOOS system and the mathematics underlying the system algorithms. A major portion of this document is reserved for a user's manual (providing detailed information relating to the operation of all system options) and for MOOS program documentation.

PREFACE

This is Volume I of a two-volume Final Report by Pattern Analysis and Recognition Corporation, 228 W. Dominick Street, Rome New York and represents work performed under Contracts F30602-75-C-0226, F30602-73-C-0352, and F30602-73-C-0351, Job Order Numbers 55971309, 55971306, and 55971305 for the Rome Air Development Center, Griffiss Air Force Base, New York. Mr. John C. Faust and Mrs. Patricia J. Baskinger were the RADC Project Engineers.

Evaluation

During the past seventeen years, RADC has conducted an exploratory development program to establish techniques for digital signal processing and pattern recognition. It has become evident to us that the solution to the pattern recognition problem does not lie wholly in learning machines, statistical approaches, heuristic programming, formal linguistic approaches or any other single model or technique. Hence, we adopted an interactive approach to the solution of pattern recognition problems, coupling a knowledgeable human problem-solver with an interactive computer graphics system. The general purpose computer contains a library of data analysis, digital signal processing and pattern classification algorithms. By means of the graphics display console, a human operator can analyze his data, and based on what he sees coupled with any a priori knowledge he may possess, choose an appropriate signal processing/pattern classification procedure, observe the results and continue to iterate in this manner. Eventually one of two things will happen: (1) he achieves an acceptable level of performance, whereby the output of the computer consists of the design parameters for a signal processor/classifier which can be implemented by means of special purpose hardware or software, or (2) he reaches a point where no further improvement seems possible. In this case, he has hopefully gained insight into the reasons why an acceptable level of performance was not achieved.

The physical realization of this interactive approach is the RADC Pattern Recognition Design Facility which has two major elements: an interactive system for waveform data analysis and feature extraction, entitled Waveform Processing System (WPS) and an interactive system for vector data analysis and pattern classification, entitled the On-Line Pattern Analysis and Recognition System (OLPARS). In addition, the facility has a hybrid computer for analog preprocessing as well as an analog to digital conversion capability.

WPS is being implemented on the PDP-11/45 computer and uses a Vector General graphics terminal as the primary interactive device. The system includes its own executive software, filing system, display package, mathematical transform package, and feature extraction language. The input to the system is in the form of digitized waveform data. The system is built as a series of overlays which are callable by the operator from a menu which is displayed on the CRT. The data, in the form of data trees, is available to the analyst by means of the interactive devices on the Vector General console.

OLPARS is resident on two systems. One version is on the PDP-11/45 computer under WPS. This is a single-user system employing high performance interactive graphics, and, as a module under WPS, provides for ease of interaction between the feature hypothesis mode conducted under WPS, and rapid testing of these hypotheses under OLPARS. However, since this system is built on a mini-computer there are core limitations in terms of the size of the data base which can be processed.

A second version of OLPARS is implemented on the HIS 6180 computer under the MULTICS operating system. (It is this version which is documented in this report.) MULTICS is a time-sharing system that utilizes a virtual memory concept. Interactive graphics capability is provided by a Tektronix 4002A storage tube with alphanumeric keyboard, joystick, and hardcopy unit. MULTICS/OLPARS has a distinct advantage over the PDP-11/45 OLPARS in terms of storage capacity, ease of data access, multi-user environment, and data base sharing among users. Besides providing more advanced pattern classifier logic design capability, the system is available to other Government agencies and their defense industry contractors by remote access through the ARPA computer network.

Both versions of OLPARS include their own executive software, filing system, display package, and software modules for feature evaluation, vector data structure analysis, measurement transformation, and classifier logic design. In general, OLPARS requires that the input data consist of 100 or fewer digital measurements (100-dimensional vectors).

The RADC Pattern Recognition System Design Facility provides the Air Force with a powerful capability, unique within DoD, for solving a wide range of target identification problems in the areas of command, control, communications and intelligence. Concurrent with the development of the Facility, it has been applied to real-world problems involving the design of classification logic. Feasibility of the Facility to solve the following problems has been established: photometric and radar satellite signature identification, ground sensor target classification, ELINT emitter identification, land type classification for automated cartography, speech recognition and handprinted alphanumeric character recognition.

Patricia J. Baskinger
PATRICIA J. BASKINGER
Project Engineer

TABLE OF CONTENTS

VOLUME I

SECTION 1

MOOS - MULTICS OLPARS Operating System

1.1 INTRODUCTION

The development of interactive graphics computer systems for use in the detection, identification, and transformation of patterns contained in high-dimensional data has been a continuing program at Rome Air Development Center since 1968 (RADC-TR-70-139; RADC-TR-71-177; RADC-TR-72-71; RADC-TR-73-241). This long-standing effort has resulted in the implementation of OLPARS (the On-Line Pattern Analysis and Recognition System), IFES (the Image Feature Extraction System), and WPS (the Waveform Processing System). This report contains detailed design and user-oriented information related to MOOS (the MULTICS OLPARS Operating System), an advanced version of OLPARS currently resident upon the Honeywell 6180 MULTICS computer system. The currently operational system represents an implemented version of the operations described in a previous report (RADC-TR-73-241); appropriate sections of that report are retained within this document. This report contains brief descriptions of the MOOS system and the mathematics underlying the system algorithms. A major portion of this document is reserved for a user's manual (providing detailed information relating to the operation of all system options) and for MOOS program documentation.

The reader is referred to articles by Sammon [2], Kanal [1], and Simmons [3] for more detailed explanations of the rationale and philosophy underlying computer-based interactive pattern analysis and recognition systems. Briefly, it has been noted that no particular solution (among a choice of learning machines, statistical approaches, spatial filtering, heuristic programming, or formal linguistic approaches) has proved relevant to all pattern recognition problems. MOOS provides a selection of modular software approaches which are applicable to a given set of data to be analyzed in an interactive setting. Provisions are available for feature evaluation, redefinition, and pattern classification schemes which provide swift feedback to the data analyst. Modification of operational parameters may then be tried on-line in an attempt to compute the optimal solution to the relevant problem. Kanal [1] has listed several features which are desirable in an interactive system for mathematical or graphical representations. In each case, MOOS represents an expansion and improvement of the OLPARS design. They are:

- o Simple procedures for system control and communication, i.e., a display of relevant options in the form of a menu on the side of the graphics display, and a selection of options and parameter specifications through a simple language using the alphanumeric keyboard.

- Response in an on-line mode allowing for rapid formulation, insertion, and testing of alternate hypotheses relating to data set structure and logic design.

- The ability to select, label, merge, and split data sets on-line; to perform set operations on data sets and subsets; and to retrieve selected data and trial test algorithms and designed solutions.

- The ability to select with minimal delay any option in the system available for execution.

- The ability to temporarily store and compare results of various algorithms on a data set.

- The ability to obtain intermediate results while sequencing through various operations.

- A requirement for swift generation and modification of system algcrithms and programs.

- Storage of large quantities of data without slowing system operations; provisions for dynamically accessing multiple data sets and subsets for application by system algorithms or for system display while maintaining a swift execution of system subroutines.

This report is devoted primarily to a data analyst's view of the capabilities within MOOS to produce a feasible solution to a problem. The remainder of Section 1 contains an overview of the structure of MOOS and of the general capabilities provided the user, as well as brief discussions of the computational algorithms utilized within the system. Section 2 is a user's manual for MOOS; it assumes a working knowledge of the basic system capabilities, that is, information is provided on the mechanics of manipulating data without there being provided specific guidance to operations which might be useful for a specific problem. Finally, Sections 3 and 4 contain complete system file descriptions and program documentation.

1.2 A Functional Overview of MOOS

MOOS retains the functional outline of OLPARS, yet has expanded or added features to each of the modules consistent with experience gained during implementation and practical use of both systems. This section consists of a description of the functional organization of MOOS, and of brief descriptions of the various system operations. Detailed mathematical outlines of the major system computations are contained in Section 1.3.

The pattern recognition problem is described as the recognition of the state of an environment based on L measurements or features extracted from the environment. Thus, the pattern recognition problem is composed of feature extraction, that is, the definition of the measurements, and of pattern classification. The objective in selecting features is to provide a set of measurements which yield information which will aid in discriminating between the various environmental states. The pattern classification problem requires that we design the recognition logic, which classifies the state of the environment using the previously defined L features.

The concept of a vector space is fundamental to all of the problems discussed here. The features (measurements) define the basis of the space; an object or an event is represented as a vector in that space. Feature extraction involves defining the representation space, and pattern classification involves defining the partitionment of this space into regions associated with each of the states (or classes) of the environment. In order to solve a pattern classification problem, statistical sample vectors from each state (or class) must be collected and analyzed to yield a satisfactory classification logic.

The pattern analysis problem differs from the pattern classification problem in that the states (or classes) of the environment are a priori unknown to the researcher. The data comprises a set of L-dimensional vectors which must be analyzed to determine the natural or inherent classes contained in the vector data. The detection and identification of a substructure of clusters (sample vectors which cluster together in the vector space) is the solution to this problem.

The vector data structure is represented within MOOS as a hierarchical tree where each node corresponds to a list of vectors. Partitionment of a list of vectors (node) is represented by branches to lower-order nodes emanating from the node corresponding to the original list, with each subnode being associated with a sub-list. Suppose, for example, that we have collected statistical sample vectors from K classes and wish to design a decision logic which adequately discriminates between data from these classes. Initially, at the time when the vector data from each of the K classes are loaded into the system, the data tree would appear as shown in Figure 1-1 where the names $C_1, C_2, \ldots, C_K$ correspond to the K data classes. The MOOS user can choose to process the data associated with any node(s). Throughout the entire system the concept of a "current data set" is used. This refers to the data that the on-line user has most recently designated for processing. The set could contain the data under a single node of a preselected tree, or it could contain all of the data associated with an entire tree. In either case, the "current data set" is related to only one data tree. If the MOOS user chooses to invoke a transformation or a clustering option, the data tree structure will be modified to reflect the resulting change. Suppose that the system user decides to discard a subset of the L original measurements (features) based upon the outcome of a measurement evaluation. This action is easily accomplished using the appropriate linear transformation, which would alter the data structure by producing a new tree as shown in Figure 1-2. At this point, the user could choose either the transformed data or the original data for further processing. Notice that when the transformation is applied at the topmost node of a tree, the structure below the node is maintained and the transformation is applied to all the data vectors. A transformation may be selectively applied to the data below a specified node, in which case a new tree is generated, involving only the data corresponding to the selected node.

As a result of using any one of the many OLPARS data analysis techniques, the user may wish to restructure the data into clustered subsets. Clustered subclasses are represented by subnodes under the node corresponding to the parent class. Suppose that class C_1 of the data represented in Figure 1-1 was subdivided via on-line analysis into the subclasses labeled C_{1a} and C_{1b}. The resulting data structure would be as shown in Figure 1-3. Notice that

$$C_{1a} \cup C_{1b} = C_1, \qquad C_{1a} \cap C_{1b} = \emptyset.$$

The MOOS facilities for solving problems of pattern analysis and classification consist of the following types of routines (individual option names are underlined - refer to Section 2 for descriptions of those programs; section number references point to related sections):

<u>Data</u> <u>Input</u>, <u>Storage</u> <u>and</u> <u>Output</u>

- o Data input from cards (<u>crdinput</u>), tape (<u>tapinput</u>) and other MULTICS files (<u>fileinput</u>, <u>restore</u>, <u>restorec</u>).

- o Permanent storage facilities in which MOOS data may be maintained either for the exclusive access of a given user (exclusive user storage) or for common access by a number of analysts (common user storage).

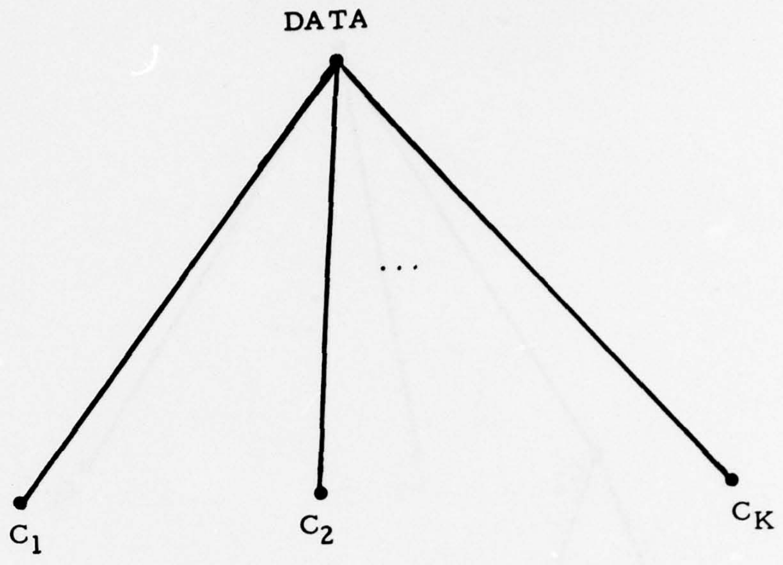


Figure 1-1. Initial Data Tree

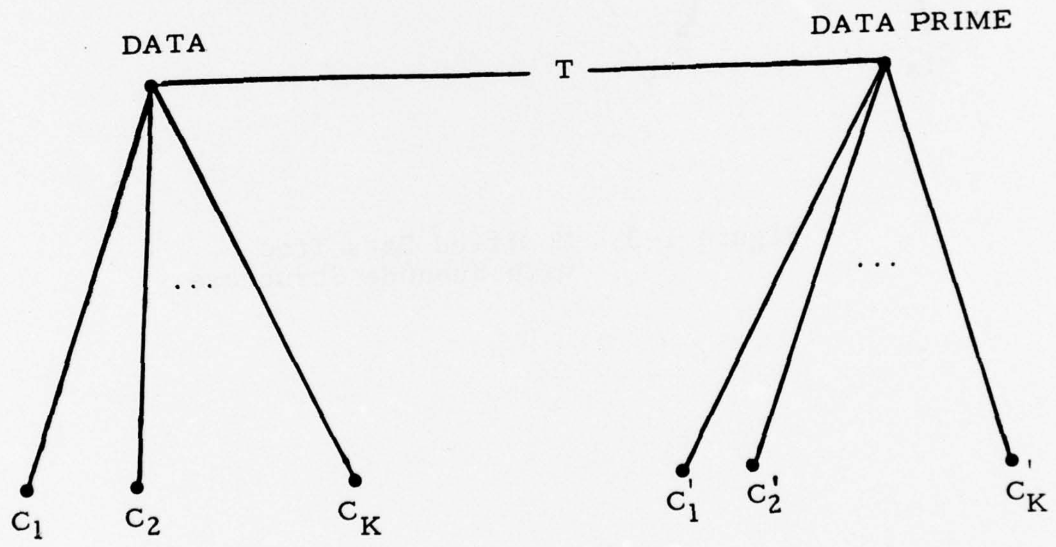


Figure 1-2. Transformed Data Tree

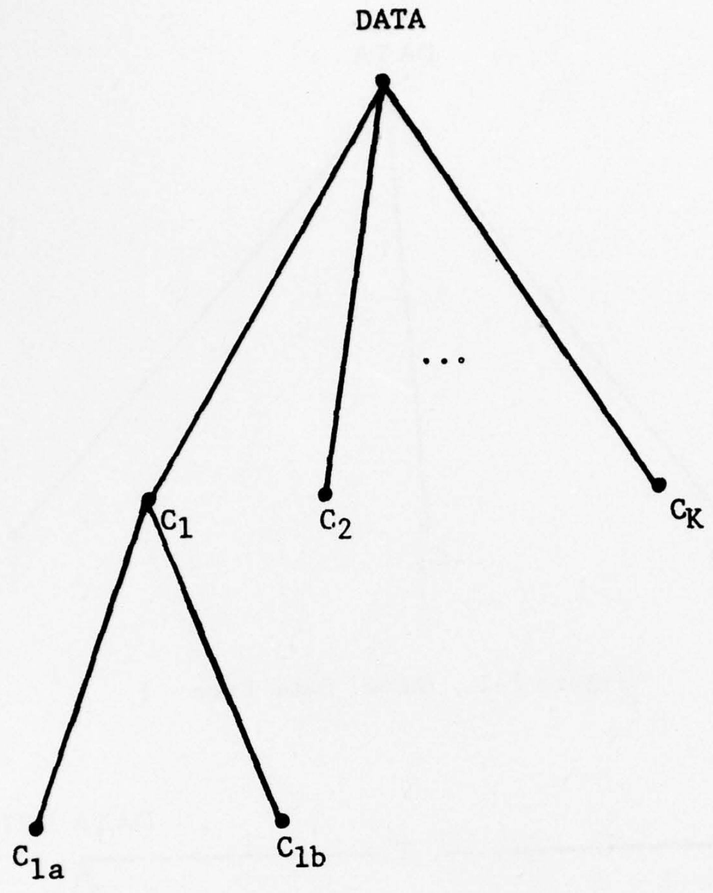


Figure 1-3: Modified Data Tree with Subnode Structure

Data trees may be output to either type of storage area (save, savec), retrieved (restore, restorec), and deleted (cleartree, remtree). Lists of the data trees in each area may be obtained (list_ust, list_cst) by user command. In addition, MOOS logic (log$_ _ _) and projection vectors (vec$_ _ _) may be stored, retrieved and deleted from exclusive user storage, and lists of those data may be obtained separately.

o Current data storage facilities provide for immediate access to any of up to 20 data trees by standardized parameters that can be added to any MOOS function call. Programs for listing (treelist) and deleting (deletree) data trees from current storage are available. In addition, data trees within the current storage area may be modified by adding data classes from other data trees (append), by combining classes within the data tree (comnod), by deleting any data class (deletnod) or data class substructure (dsubstrc), or by removing individual data vectors from the data set (dvectors). A node substructure may be added to the current data tree via the structure analysis module (lingpart, restruct). Any data tree or data class name may be changed (chngname), and a display of the current data tree is immediately available (treedraw). Finally, a new tree may be created from data classes existing in numerous available data trees (creatree), or by extracting a percentage of data vectors from an existing data tree (crrandts). The purpose of this final option is to provide a facility for the creation of randomly assigned design and test sets for the design and independent testing of classification logic.

o Data trees from current data storage may be permanently stored on magnetic tape (tapeoput).

o The standard system limit on data dimensionality for MOOS routines is 100. However, through use of the excess measurement mode feature, data sets with a greater number of measurements may be handled. The allowable options under the excess measurement mode assist in finding a suitable subspace with 100 or less dimensions, and transform a data set to this subspace so that normal processing may begin.

The excess measurement mode is entered any time an attempt is made to enter a data set with greater than 100 dimensions, or through use of measxfrm. The chief difference between data trees handled by the excess measurement mode and normal MOOS data trees is that mean vectors and covariance matrices for each class are not stored. Once the dimensionality of an excess measurement mode tree has been reduced to 100 or less, the mean vectors and covariance matrices may be calculated by function <u>moosmode</u>. Normal MOOS operations may begin only after <u>moosmode</u> has been invoked.

An arbitrary limit of 250 dimensions has been set on excess measurement mode operations. This limit may be exceeded only through modification to certain system routines (see Section 4.2).

Programs which are allowed in the excess measurement environment include:

crdv$sal & sa2	measxfrm	tapeoput
creatree	moosmode	tapinput
dataprnt	probconf	treelist
dscrmeas	sense	trnsform

Any programs which appear in the option lists of the above functions (<u>clprint</u>, <u>trnsform</u>, etc.) may also be used while in the excess measurement mode.

<u>Data Display - Projection Planes and Display Formats</u>

o Four data display formats are provided (Section 2 introduction) for seven sets of data projection axes (<u>arbv</u>, <u>ardg</u>, <u>asdg</u>, <u>crdv</u>, <u>eigv</u>, <u>fshp</u>, <u>gndv</u>; see Section 1.3). Facilities for user manipulation of these data projection displays include printouts (<u>hgprint</u>, <u>clprint</u>), indexing specified points (<u>index</u>), modifying scale factors (<u>scale</u>), sequencing appropriate data projections (<u>seq</u>), storing projection vectors for later use (<u>vec$save</u>), changing the data class composition of a display (<u>elimclas</u>) or highlighting specified data classes (<u>intensfy</u> - a bargraph plot for one-space displays only), and implementing partitions drawn via cursor on the display terminal (<u>dra$bndy</u>, <u>dboundry</u>, <u>redraw</u>).

Measurement Evaluation

o Three measurement evaluation computations (dscrmeas, probconf, features; see Section 1.3) are provided the MOOS user. Rank order displays have been implemented; manipulations available for these displays include printout (hrdcpy), rankings for selected classes, class pairs, or measurements (rnk$_ _ _), display of the distribution of data along a selected measurement in histogram format (histgram), and selection of a measurement subset for inclusion within a data set of reduced dimensionality (sel$_ _ _, un$_ _ _). Finally, a program for data set reduction is operational (trnsform).

Data Tree Transformations

o Three additional options are available for creation of transformed data sets: normalization (normxfrm), eigenvector transformation (eigentrn), and linguistic transformation of individual measurements (measxfrm).

Structure Analysis Partitions and Projections

o The creation of subnode structure in a data tree (the structure analysis function) can be implemented via partition of a data projection display (restruct following use of dra$bndy on any of the data projection displays) or by linguistic statements (lingpart) based on a priori knowledge of the ranges and relationships of data distributions within and between data classes. Linguistic partition allows development of data partitions via logical statements composed of any existing measurement threshold or legal arithmetic combination of measurements, thresholds, or variables using PL/1 conventions.

o An additional data projection display is available for the structure analysis function in the form of a nonlinear mapping algorithm (nlm; see Section 1.3). This algorithm has been equipped with a data set clustering algorithm which allows its use on large data sets despite the time and space limitations inherent in the maintenance of arrays related exponentially to data set size, which are required by this algorithm.

Classification Logic Design and Evaluation

The MOOS Logic Design (LD) facilities provide extensive mathematical/graphical techniques for allowing the user to tailor decision logic design to the structure of the class data. In general, pattern classification is undertaken following a pattern analysis conducted on each of the data classes for which logic is to be designed. The purpose of this analysis is to ensure that each data class is unimodal; that is, the vectors from each class are clustered in one region of the measurement space. Although not always required, the unimodality property is highly desirable in order to ensure an effective logic design. In those cases where the class data is found to be multimodal, our philosophy dictates that each mode be identified and the sample vectors corresponding to each mode be grouped as a named subclass. Upon completion of the logic design, the decision region in the measurement space corresponding to each subclass can be reidentified with the original multimodal classes (reasname).

Figure 1-4 presents a functional overview of the logic design facilities. Upon selection of an LD option, a logic tree is initialized by the system with a single node consisting of all the lowest-order data classes in the current data set (there is no requirement that the current data set be the senior node in any data tree).

The system will keep a record of the decision logic as it is being designed. The actual form of the logic constructed in this manner will be that of a hierarchical tree (draw), where each node corresponds to a partial decision. For example, suppose that there are five classes (K=5), labeled B, C, F, S, and T, and the user first separates B, C and F, from S and T using a between-group projection described below. At this point, the system would store the piecewise linear logic and represent the current logic as the tree shown in Fig. 1-5. Next, the user could choose to work with the group B, C, and F. Suppose that he selects the eigenvector projection corresponding to this data. Further, let us suppose that he constructs another piecewise linear logic to discriminate B and C from F. The decision logic tree would then appear as shown in Fig. 1-6. Suppose that the user could not adequately discriminate between B and C or between S and T using the eigenvector method, and therefore completed the within-group discriminations using the Fisher pairwise discriminant technique.

The basic idea behind this interactive design technique is that between-group logic will be used to design the partial logic for nonoverlapping classes, whereas the complete within-group logic computations will be used for statistically overlapped classes.

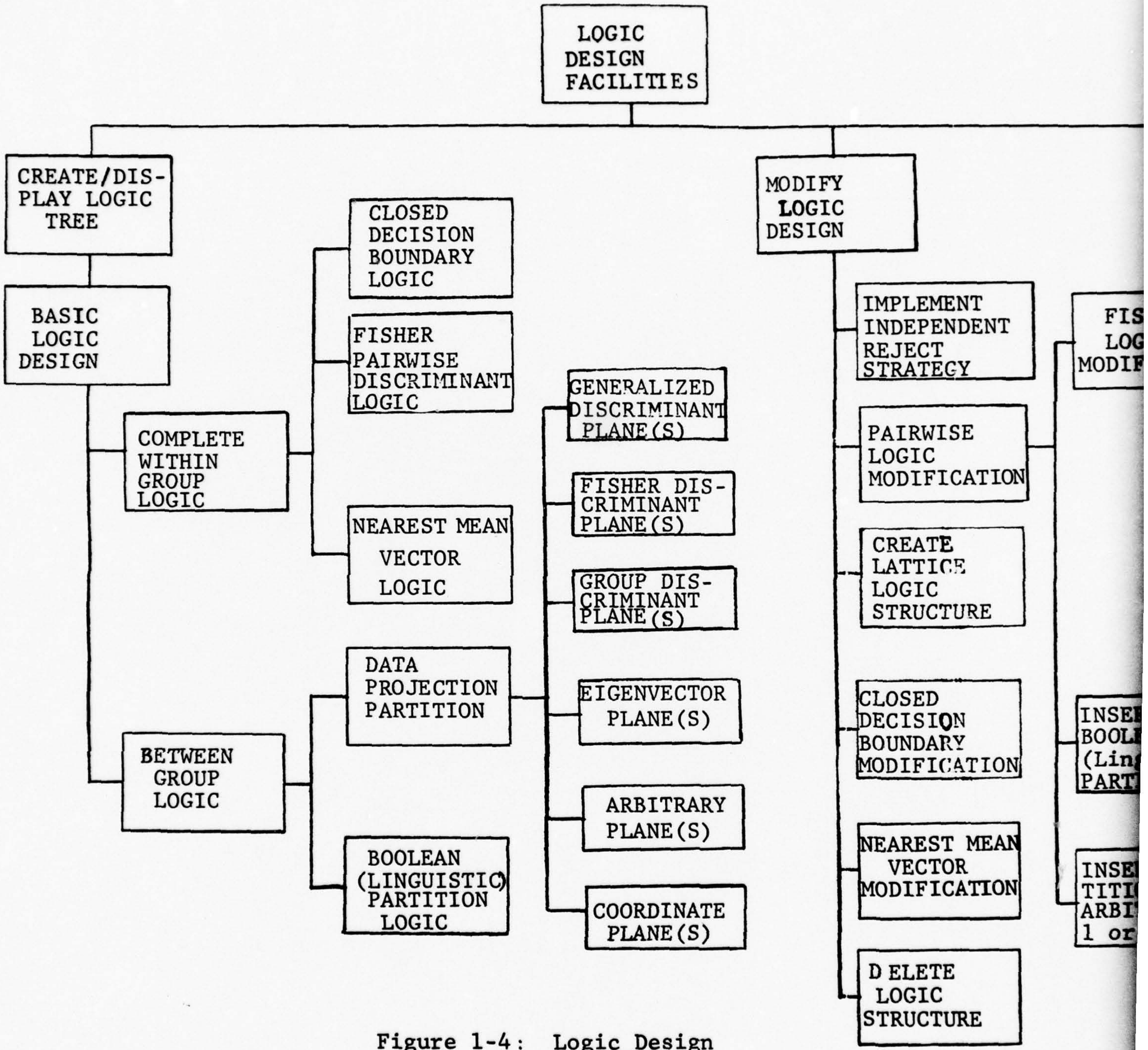
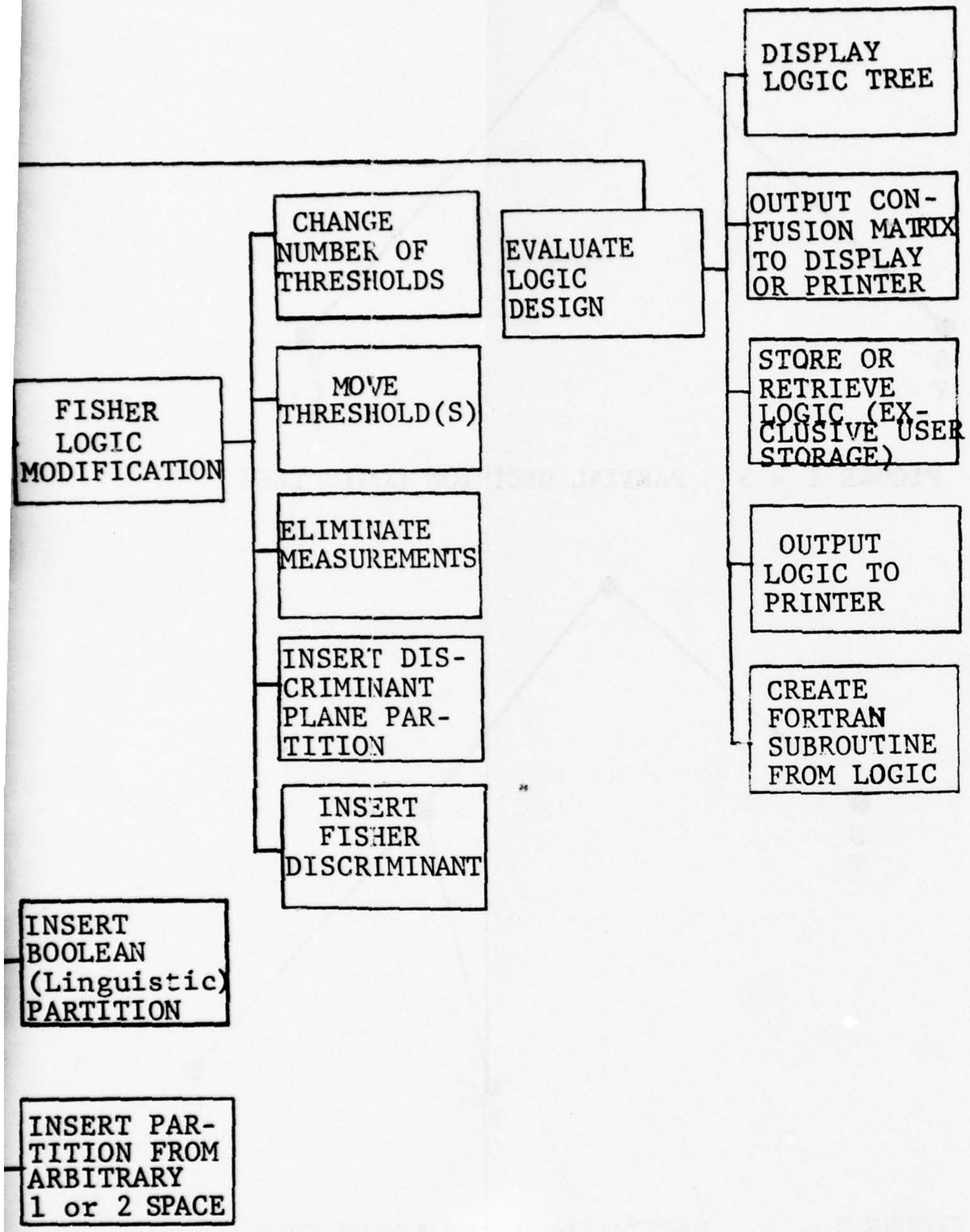

LOGIC DESIGN FACILITIES
CREATE/DISPLAY LOGIC TREE
BASIC LOGIC DESIGN
COMPLETE WITHIN GROUP LOGIC
BETWEEN GROUP LOGIC
CLOSED DECISION BOUNDARY LOGIC
FISHER PAIRWISE DISCRIMINANT LOGIC
NEAREST MEAN VECTOR LOGIC
DATA PROJECTION PARTITION
BOOLEAN (LINGUISTIC) PARTITION LOGIC
GENERALIZED DISCRIMINANT PLANE(S)
FISHER DISCRIMINANT PLANE(S)
GROUP DISCRIMINANT PLANE(S)
EIGENVECTOR PLANE(S)
ARBITRARY PLANE(S)
COORDINATE PLANE(S)
MODIFY LOGIC DESIGN
IMPLEMENT INDEPENDENT REJECT STRATEGY
PAIRWISE LOGIC MODIFICATION
CREATE LATTICE LOGIC STRUCTURE
CLOSED DECISION BOUNDARY MODIFICATION
NEAREST MEAN VECTOR MODIFICATION
DELETE LOGIC STRUCTURE
FIS LOG MODIF
INSE BOOL (Lin PART
INSE TITI ARBI 1 or

Figure 1-4: Logic Design Facilities


DISPLAY LOGIC TREE
CHANGE NUMBER OF THRESHOLDS
EVALUATE LOGIC DESIGN
OUTPUT CON-FUSION MATRIX TO DISPLAY OR PRINTER
STORE OR RETRIEVE LOGIC (EX-CLUSIVE USER STORAGE)
FISHER LOGIC MODIFICATION
MOVE THRESHOLD(S)
ELIMINATE MEASUREMENTS
OUTPUT LOGIC TO PRINTER
INSERT DIS-CRIMINANT PLANE PAR-TITION
CREATE FORTRAN SUBROUTINE FROM LOGIC
INSERT FISHER DISCRIMINANT
INSERT BOOLEAN (Linguistic) PARTITION
INSERT PAR-TITION FROM ARBITRARY 1 or 2 SPACE

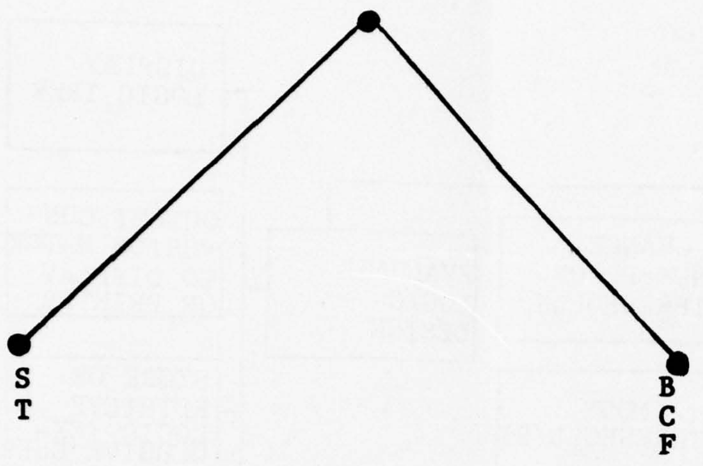


FIGURE 1 - 5: PARTIAL DECISION LOGIC TREE.

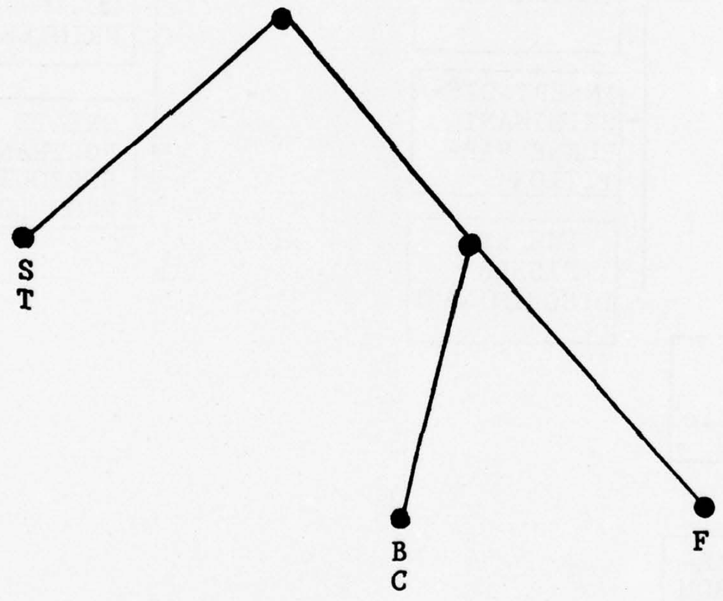


FIGURE 1 - 6: PARTIAL DECISION LOGIC TREE.

Basic logic design operations fall into two categories:

1) Logic capable of completely classifying vectors within the reference group of data classes (complete within-group logic); and 2) logic capable of identifying and partitioning completely disjoint data class groups (between group logic).

<u>Complete Within-Group Logic</u>

- o Nearest mean vector (<u>nmv</u>) logic implementation provides capabilities for classification of data utilizing one of three metrics (Euclidean distance, weighted vector distance and Mahalanobis weighted distance; see Section 1.3.3.2.1). An unknown vector, then, is assigned to the reference class for which the decision metric is minimized.

- o Fisher pairwise discriminant logic (<u>fisher</u>) is constructed by computing optimal linear discriminants and thresholds to distinguish between every pair of classes (subclasses) within a designated group. The linear discriminant is the Fisher linear discriminant given by

$$d_{ij} = \alpha W_{ij}^{-1} \triangle_{ij}$$

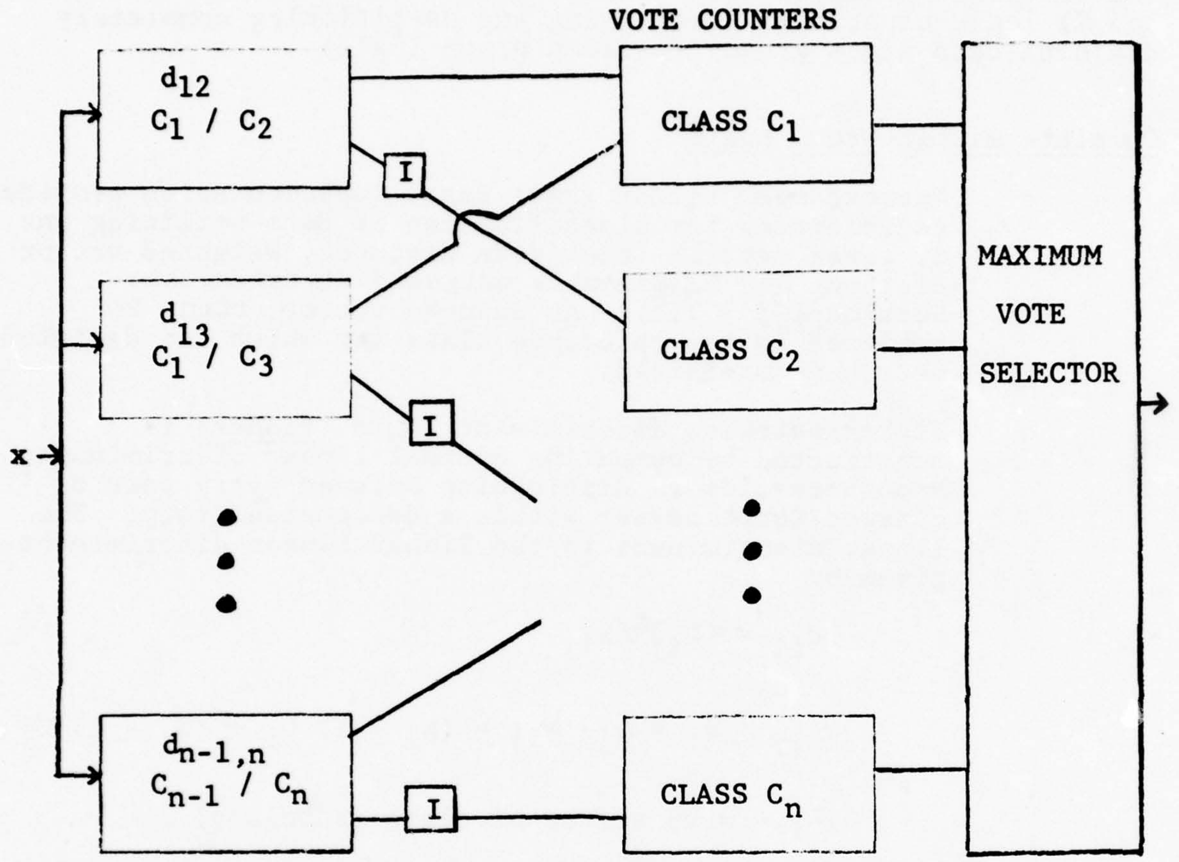
$$\triangle_{ij} = \mu_i - \mu_j : W_{ij} = (N_i - 1)\, C_i + (N_j - 1)\, C_j$$

μ_i = mean vector of class (subclass) i

C_i = covariance matrix for class (subclass) i

Once the within-group pairwise discrimination is complete, the pairwise decisions are combined to produce a final decision. The group of classes (subclasses) might be the original K classes (subclasses) of the "current data set," or the group might be composed of a subset of K. In the case where the user does not subdivide the K classes (subclasses), he would compute K $[(K - 1)/2]$ pairwise discriminants. The resultant logic for this example is shown in Fig. 1 - 7.

The output from the d_{ij} box is either a zero or a one, depending on the following criterion:



I = Inverter

FIGURE 1 - 7: PAIRWISE LOGIC

$$d_{ij}(X) = \begin{cases} 1 & \langle d_{ij}, X \rangle - \theta_{ij} \geq 0 \\ 0 & \text{otherwise.} \end{cases}$$

The signed direction ($\pm$) of d_{ij} and the threshold θ_{ij} are selected so that an output of "one" is interpreted as a vote for class i and an output of "zero" as a vote for class j. The I box is an inverter which produces a "one" out given a "zero" in, or a "zero" out given a "one" in. The votes for each class are collected and a final decision is made according to the class with the most votes.

Under the Fisher discriminant logic option, the user can select 1, 2, 3, or 4 threshold options which result in the different boundaries shown in Figure 1-8.

In Figure 1-8, μ_i is the estimated mean of class i projected onto the discriminant direction $\underset{\sim}{d}$, i.e.,

$$\mu_i = \underset{\sim}{d}^T \underset{\sim}{\overline{X}}_i \qquad i = 1, 2$$

MOOS will automatically set:

$$\theta_1 = \mu_2 - \Delta/2$$
$$\theta_2 = \mu_2 + \Delta/3$$
$$\theta_3 = \mu_2 + \Delta/2$$
$$\theta_4 = \mu_1 - \Delta/3$$
$$\theta_5 = \mu_1 + \Delta/2$$

where $\Delta = \mu_1 - \mu_2$. The regions $[\theta_2, \theta_4]$, $(-\infty, \theta_1]$, and $[\theta_5, \infty)$ are for rejects.

Fisher pairwise logic may be treated as a between-group logic through use of the <u>pairmod</u> function. In this case, each of the classes C_1, ...Cn consists of one or more of the original K classes. Further logic may then be designed to complete the classification logic.

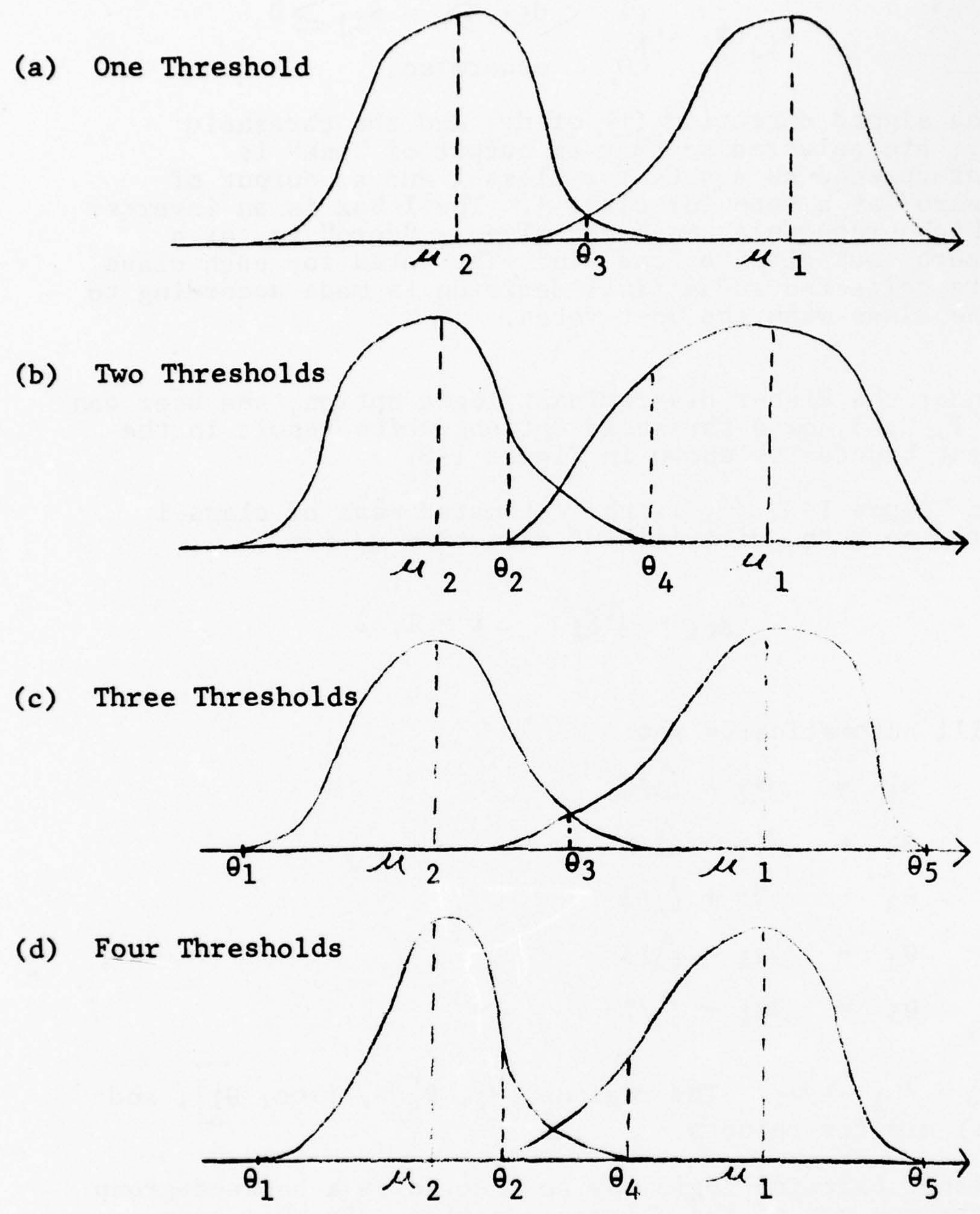


FIGURE 1 - 8: THRESHOLD OPTIONS FOR FISHER DISCRIMINANTS

o Closed decision boundary logic (closedcn) creates an L-dimensional closed hyperregion for each class of the selected data set. An unknown vector is assigned to a class if and only if it lies in the hyperregion associated with that class and no other. If an unknown vector should fall into more than one hyperregion, it may be rejected or placed in a new data tree for further logic design at the user's discretion. Vectors which do not lie within any hyperregion are rejected. (In a real environment, closed decision boundary logic would tend to reject vectors which do not belong to the set of classes intended for classification.) The following diagram (Figure 1-9) illustrates the implementation of closed decision boundary logic for two dimensions and three classes: A, B, and C.

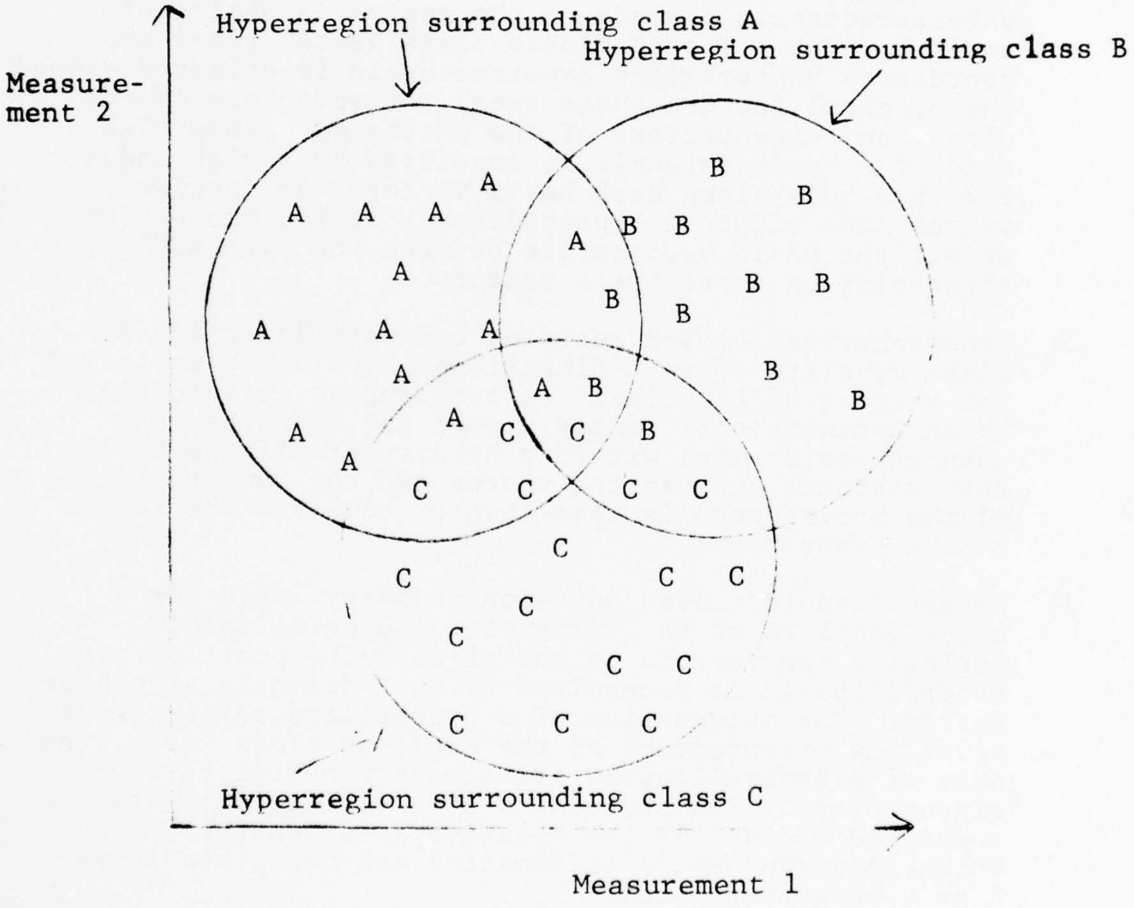

Hyperregion surrounding class A
Hyperregion surrounding class B
Measurement 2
Hyperregion surrounding class C
Measurement 1

FIGURE 1-9

Vectors which fall in the regions $A \cap (\overline{B \cup C})$, $B \cap (\overline{A \cup C})$, and $C \cap (\overline{A \cup B})$ are assigned to classes A, B, or C respectively.

Vectors which fall in the region $\overline{A \cup B \cup C}$ are rejected.

Vectors which fall in the region $(A \cap B) \cup (A \cap C) \cup (B \cap C)$ are considered "overlap" vectors and may be handled as described above.

Three types of hyperregion are available:

1) Hyperrectangular closed decision boundary logic for a class consists of an L-dimensional hyperrectangle enclosing the vectors of the class. The orientation of a hyperrectangle depends on the analyst's choice of basis vectors. The available basis vector types are: coordinate vectors (the hyperrectangle is oriented along the original feature space axes), eigenvectors of the class, and eigenvectors of the entire data set. The size of a hyperrectangle is specified by a high and a low threshold along each basis vector. An unknown vector lies within a hyperrectangle if its projections on all the basis vectors lie between the high and low thresholds on those basis vectors.

2) Hyperspherical closed decision boundary logic for a class consists of an L-dimensional hypershere enclosing the vectors of the class. A hypersphere is specified by an L-dimensional center vector and a radius. An unknown vector lies within a hypersphere if the Euclidean distance between the vector and the center vector of the hypersphere is less than or equal to the radius of the hypersphere.

3) Hyperellipsoid closed decision boundary logic for a class consists of an L-dimensional hyperellipsoid enclosing the vectors of the class. The position of a hyperellipsoid is determined by an L-dimensional center vector. The orientation of a hyperellipsoid is always along the eigenvectors of the enclosed class, i.e., the axes of a hyperellipsoid are always parallel to the eigenvectors. The shape of a hyperellipsoid may be specified by varying the relative axis lengths. Orientation and shape information are contained in an L by L weighting matrix.

   An unknown vector lies within a given hyperellipsoid if the following condition is met:

$(\underset{\sim}{X}-\underset{\sim}{M})^T\ W\ (X-M) \leq C$ where:

$\underset{\sim}{X}$ = the L-dimensional unknown vector
W = the weighting matrix
$\underset{\sim}{M}$ = the center vector
C = a size parameter analogous to the radius of a hypersphere.

Specification of the center vector, axis lengths, and "C" value are all under the analyst's control.

<u>Between-Group Logic</u>

- o <u>Data Projections</u>. An obvious drawback to computing K(K-1)/2 pairwise discriminants is the potentially large number of combinations. In most problems of interest, some of the classes (subclasses) are statistically disjoint and quite easily separated from one another. If these disjoint class groups can be identified and logic can be designed to discriminate the groups, then the pairwise discrimination need only be computed for the statistically overlapped classes (subclasses) within the group. The MOOS user will not ordinarily know a priori how to group the classes (subclasses); therefore, options are provided to project the class (subclass) data onto one- or two-dimensional subspaces and display the results. If the user detects nonoverlapping groups of classes (subclasses), he can draw separating piecewise linear boundaries on the display (<u>dra$bndy</u>). These boundaries may be stored within the system as piecewise linear hyperplane boundaries which partition the original L-dimensional measurement space (<u>creatlog</u>). The user can continue this procedure by selecting one of the class groups and projecting the corresponding data onto a new two-dimensional subspace. If between-class separation is again evident, the user may again partition the original L-space with piecewise linear hyperplanes. If, due to statistical overlap, the classes (subclasses) cannot be completely separated using this procedure, it is recommended that the user complete the logic via within-group discrimination procedures (<u>nmv</u>, <u>fisher</u>, <u>closedcn</u>).

- o <u>Scatter Plot Partitions</u>. The user has the capability to draw multiple piecewise linear convex boundaries, as shown in Figures 1-10 and 1-11. The region external to the drawn boundaries may be designated as a reject region, or can be used for data class designation (Figure 1-11). In addition, it is possible that one or more data classes may be distributed over an entire scatter plot, while others are disjoint (Figure 1-12a). In this case, the user may designate those widely-distributed data classes for both sides of a logic partition, resulting in a logic tree as represented in Figure 1-12b.

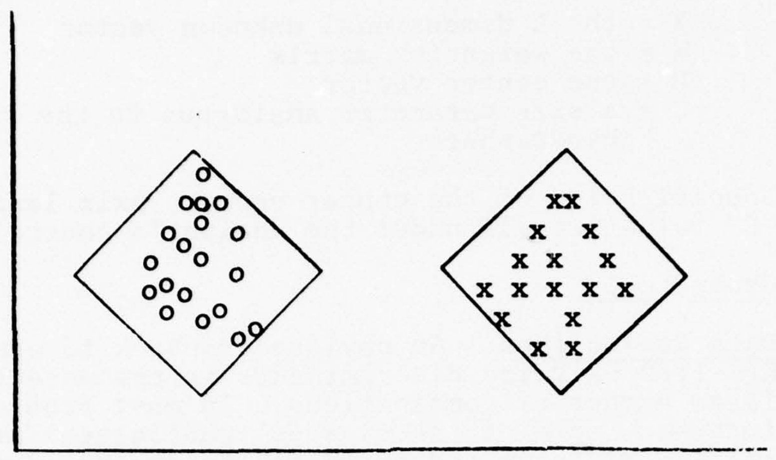


FIGURE 1 - 10: PIECEWISE BOUNDARIES WITH REJECT REGION

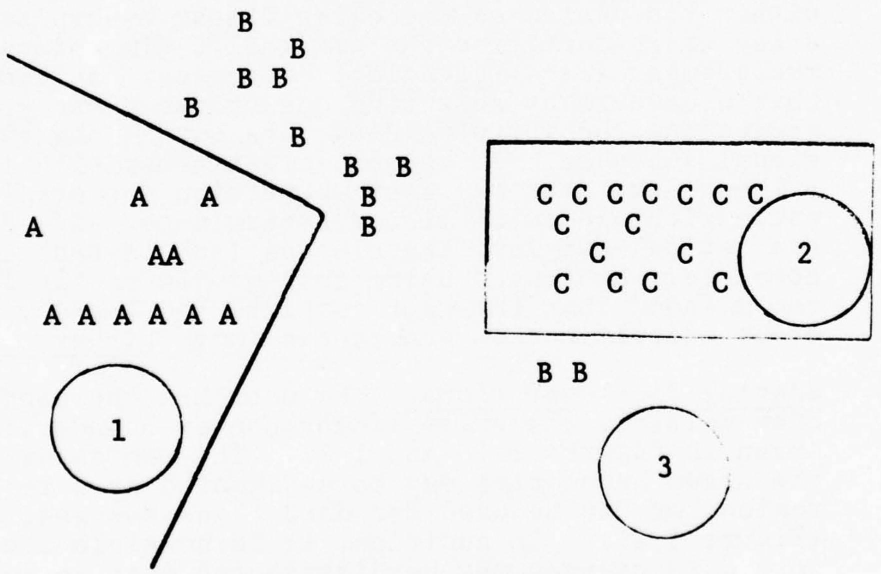
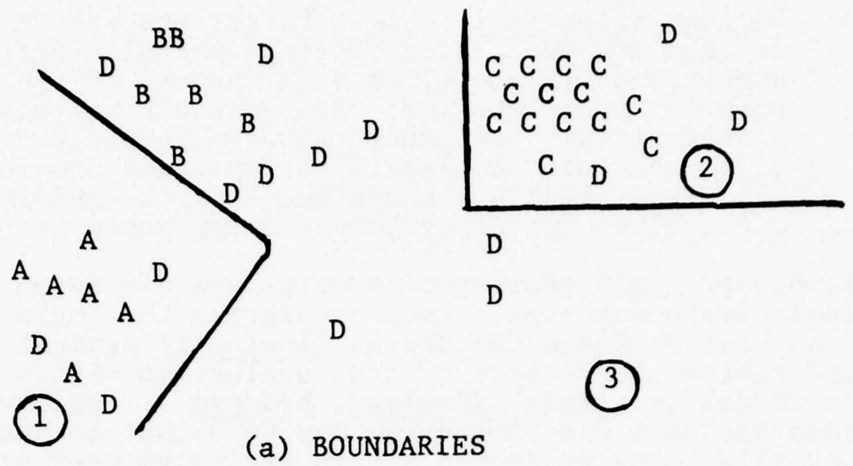
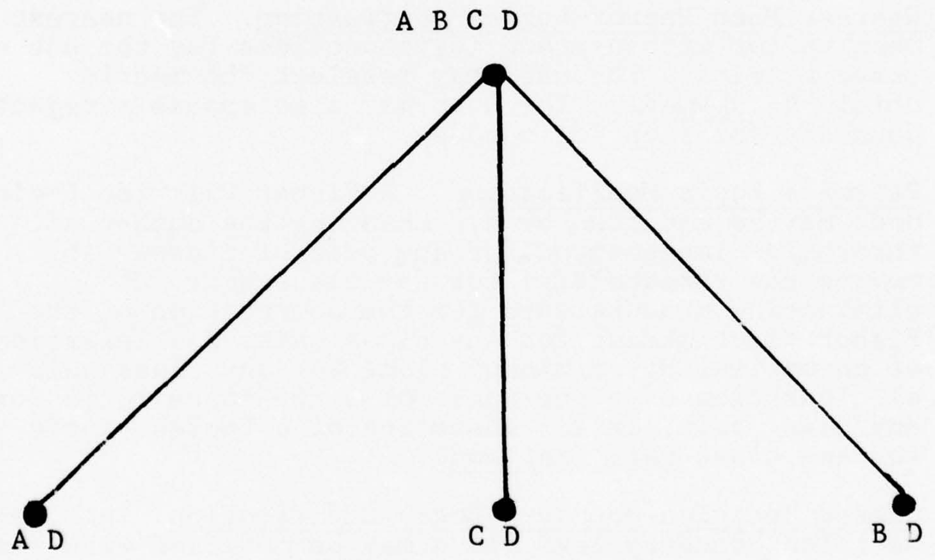


FIGURE 1 - 11: PIECEWISE BOUNDARIES WITH EXTERNAL DATA



(a) BOUNDARIES



(b) LOGIC TREE

FIGURE 1 - 12: PIECEWISE LINEAR BOUNDARIES FOR GROUP PARTITIONS

- Boolean (linguistic) Logic Partitions. MOOS provides for the implementation of linguistically-defined logic partitions (linglogc) through the use of the PL/1 compiler under MULTICS. The user can write any Boolean statement (any statement that can be evaluated true/false) for use as classification logic provided that it is a legal PL/1 statement and that it conforms to conventions for referencing vector measurements.

Temporary logic evaluation results are displayed following any logic implementation. Upon completing the logic design, the user can next evaluate the design (logicevl) against any data set and review the results of that evaluation within a confusion matrix format (summrycm, displacm, hrdcpym). Logic which provides adequate discrimination may be output to the system printer (listlogc) or stored within exclusive user storage (log$_ _ _). Inadequate logic may be supplemented (lingrjct), modified (nmvmod, pairmod, closemod), or deleted (deletlog).

- Independent Reject Strategies. Any final classification node of the logic tree may be appended with a Boolean reject strategy (lingrjct). A vector classified at a node and evaluated as false by the strategy will be rejected.

- Nearest Mean Vector Logic Modification. The nearest mean vector within-group logic provides for the use of three metrics. The user may reselect the metric utilized (nmvmod). The user may also choose a reject boundary for each logic node.

- Pairwise Logic Modification. A Fisher Pairwise Logic node may be modified by a) changing the number of thresholds implemented for any pair of classes, b) moving the threshold(s) for any class pair, c) eliminating measurements for the computation of the Fisher discriminant for any class pair, d) insertion of an optimal discriminant plane for any class pair, e) insertion of a one-space or a two-space logic for any class pair, or f) insertion of a Boolean logic for any class pair (pairmod).

- Closed Decision Boundary Logic Modification. A closed decision boundary logic node may be modified with respect to the type of hyperregion used to surround a class, or the parameters which specify a given hyperregion may be changed (closemod).

- o <u>Logic Structure Deletion</u>. Any unsatisfactory logic structure in the logic tree may be deleted (<u>deletlog</u>) and new logic implemented.

<u>Lattice Logic Structure</u>

A capability has been provided (<u>latclogc</u>) to allow the analyst to create a lattice-type logic tree structure. This allows, in effect, for two or more logic nodes in a MOOS logic tree structure to branch together. Two examples of the utility of this feature are given below:

- o <u>Duplicate Logic Tree Substructure</u>. Consider the logic tree structure in Figure <u>1-13 a)</u> for the four classes A,B,C.D. Each decision represents a between-group type logic. The classes present at logic nodes 2 and 4 are the same (A,B, and C). Rather than duplicate the logic which was created at logic node 2, the analyst may cause logic node 4 to be connected to logic node 2 (indicated by the dotted line in the figure).

- o <u>Lattice Logic Tree Structure</u>. The normal MOOS logic tree structure has been implemented with the idea that a relatively small number of decisions will be necessary to separate one set of classes or subclasses from another. The following example illustrates a pitfall of this type of logic tree structure and the remedy. Consider the logic tree structure in Figure 1-13 b). The analyst has made a number of tests (logic nodes 1, 2, 4, 6, ..., N) designed to separate class A from classes B and C. Note that many logic nodes with the classes B and C present (3,5,7,...M) were created. In order to allow the creation of only one logic tree substructure for the classification of B and C, the logic nodes where B and C are present may be connected.

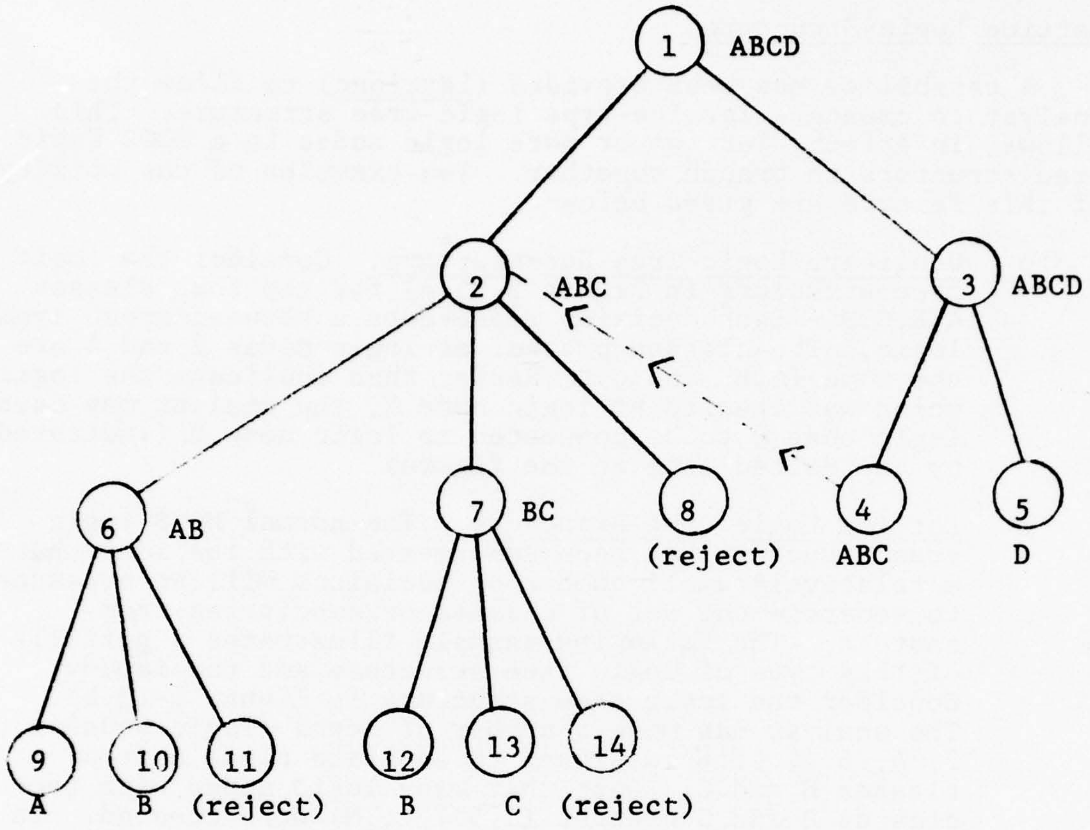

1 ABCD
2 ABC
3 ABCD
6 AB
7 BC
8 (reject)
4 ABC
5 D
9 A
10 B
11 (reject)
12 B
13 C
14 (reject)

Figure 1-13a

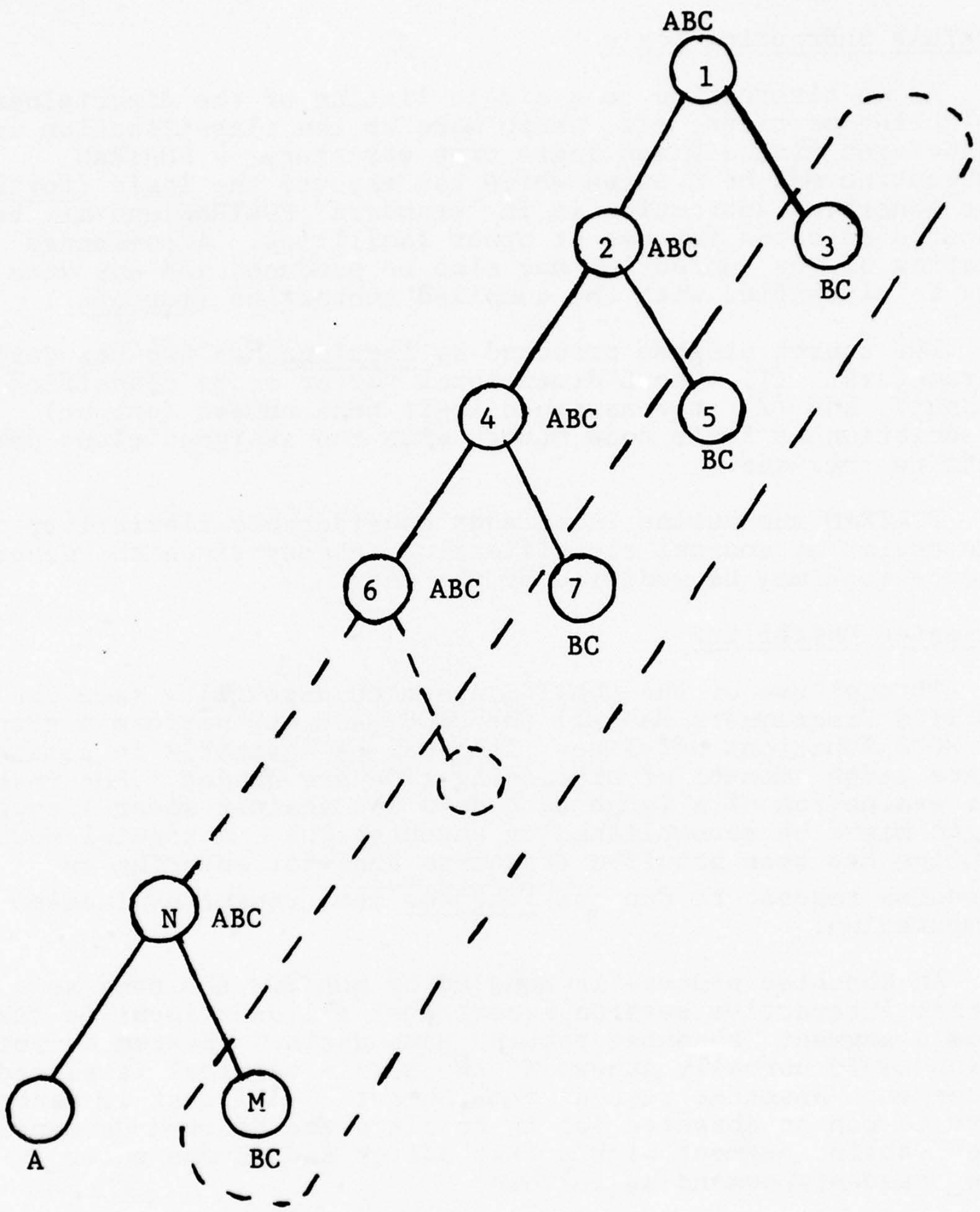


Figure 1-13b

FORTRAN Subroutine Logic

As an alternative to a simple listing of the discriminants, weighting matrices, etc. which make up the classification logic associated with a given logic tree structure, a FORTRAN subroutine may be created which can execute the logic (fortlogc). The generated subroutine is in "standard" FORTRAN and may be punched on cards for use at other facilities. A commented listing of the subroutine may also be produced and any data set may be classified with the compiled subroutine (forteval).

The source program produced by fortlogc has two standard parameters: (1) the L-dimensional vector to be classified (input), and (2) the assigned logic node number (output). The association of logic node number with the assigned class name is left to the user.

FORTRAN subroutine logic adds considerable flexibility to the design of unusual classification schemes since the generated source code may be modified by the user.

Absentee Capability

Through use of the MULTICS absentee capability (see the MULTICS Programmers Manual) the MOOS user may perform a sequence of MOOS functions off-line. This may be desirable in some cases where large amounts of processing time are needed. For instance, the evaluation of a large test data set against several types of logic might be accomplished by absentee job. A special MOOS routine has been provided (features_abs) for entering an absentee request to run the features measurement evaluation computation.

An absentee process is handled by MULTICS the same as a normal interactive session except that all user input is taken from a segment "absentee_request_name.absin." System output which would normally appear at the user's terminal is placed in a segment "absentee_request_name.absout." All that is necessary to run an absentee job is to place the desired commands in the ".absin" segment with a text editor and invoke enter_ abs_ request command as follows:

```
ear   absentee_request_name
```

The following example illustrates an absentee job which creates and evaluates fisher pairwise logic on a data set and prints the results on the line printer.

```
cwd >udd>C>OLPARS              (change working directory to
                                 OLPARS)
restore datatree               (restore the design data set)
4                              (answer questions related to
300                              console type and baud rate)
fisher                         (invoke fisher pairwise logic)
0                              (select option 0)
1                              (select 1 threshold)
8                              (set minimum vote count to 8)
yes                            (hardcopy confusion matrix and
yes                              list of errors to line printer)
no                             (halt fisher calculation)
logout
```

It is obvious that considerable familiarity with the interactive queries of MOOS functions is necessary since all queries must be correctly answered within the ".absin" segment. The absentee job in the above example could be performed on a number of data sets by simply changing the tree name "datatree" to the names of other data sets.

1.3 MOOS MATHEMATICS

This section of the report presents the mathematical justification or explanation of the algorithms used in MOOS. Only the general explanation is included here; the reader is referred to Section 3 for details concerning algorithm implementation.

1.3.1 Measurement Evaluation

In solving a pattern classification problem, the researcher will often be concerned with the discriminatory qualities of the L measurements. In general, it is desirable to use the minimum number of measurements that achieves a satisfactory solution. To this end, the MOOS system provides three (3) methods for ranking the discriminatory power of a set of L measurements.

If desired, the rankings may be used as the basis for a measurement reduction transformation to a subset consisting of the M most discriminatory measurements. An optimal method for selecting a subset of M measurements must involve a consideration of the decision logic criterion, such as the Bayes Risk or the probability of error. This, in turn, requires the estimation of the joint probability functions for all possible n-tuples. The obvious computational difficulties in obtaining an optimal ranking preclude this approach in all but the simplest problems. Therefore, the following sub-optimal algorithms are provided as options to rank-order the L measurements $x_1, x_2, \ldots, x_L$. Each algorithm provides three distinct types of rankings. The first uses a significance measure of a particular component, e.g. x_p, for discriminating class i from class j; this significance will be designated by $M_{ij}(x_p)$. The second type of ranking uses a significance measure of x_p for discriminating class i from all other classes, and is designated $M_i(x_p)$. The last type of ranking uses a measure of the overall significance of x_p for discriminating all classes, and is designated $M(x_p)$.

1.3.1.1 The Discriminant Measure

This algorithm is implemented in the MOOS function <u>dscrmeas</u>. This significance measure is particularly useful for ranking the L measurements when the class conditional probability distributions are approximately unimodal. The discriminant measure for differentiating class i from class j using measurement x_p is defined as:

$$M_{ij}(x_p) = \left[\bar{x}_p^{(i)} - \bar{x}_p^{(j)}\right]^2 \Bigg/ \left[(N_i-1)\left[\hat{\sigma}_p^{(i)}\right]^2 + (N_j-1)\left[\hat{\sigma}_p^{(j)}\right]^2\right]$$

where

$\bar{x}_p^{(j)}$ = the estimated mean of class j along measurement x_p

$\hat{\sigma}_p^{(j)}$ = the estimated standard deviation of class j along measurement x_p

N_i = the number of vectors in class i

The discriminant measure for differentiating class i from all other classes using measurement x_p is defined as:

$$M_i(x_p) = \sum_{j \neq i}^{K} M_{ij}(x_p)$$

Finally, the discriminant measure for distinguishing all classes using measurement x_p is defined as:

$$M(x_p) = \sum_{i=1}^{K} M_i(x_p) = \sum_{i=1}^{K} \sum_{j \neq 1}^{K} M_{ij}(x_p)$$

1.3.1.2 The Probability of Confusion Measure

This algorithm is implemented in the MOOS function probconf.

This measure is recommended when the assumption of class unimodality cannot be justified. It is valid for any probability distribution since it essentially measures the overlap of the class conditional probabilities. Computationally, it is much more complex than the previous measure.

Let x_p designate the measurement under evaluation and $P(x_p/C_j)$, j = 1, 2, ..., k be the marginal class conditional probability distributions. Next, consider the distributions for the two classes i and j shown in Figure 1 - 14. The measure for differentiating class i from class j using x_p is defined as follows:

$$M_{ij}(x_p) = \int_{-\infty}^{\theta_1} P(x_p/C_j)dx_p + \int_{\theta_1}^{\theta_2} P(x_p/C_i)dx_p + \int_{\theta_2}^{+\infty} P(x_p/C_j)dx_p$$

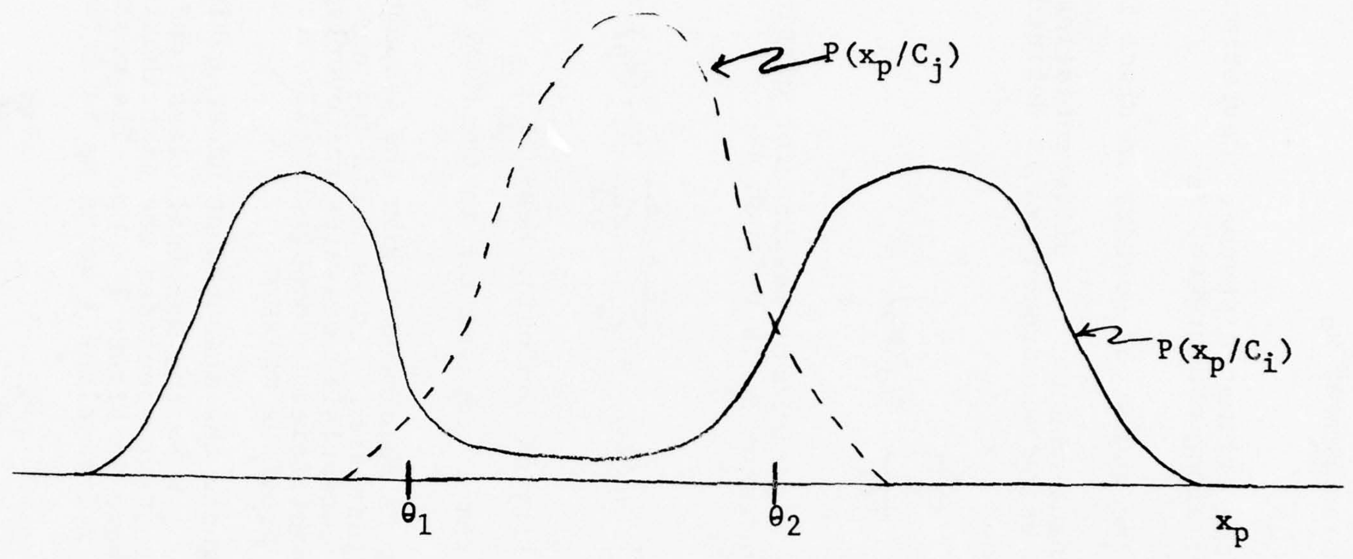


FIGURE 1 - 14

Marginal Class Conditional Probability Distributions (2 Classes)

Since the functional forms of the class conditional probabilities are not known, we estimate the marginal class distributions using the sample data. This method makes use of histogram approximations like those shown in Figure 1-15. A detailed discussion of histogram computation will be presented later.

The measurement x_p will be divided into cells of width Δ. The probability that a sample from class j will occupy the rth cell along measurement x_p is given by

$$P_{rp}^{(j)} = \int_{r^{th}\ \mathrm{cell}} P(x_p/C_j)\,dx_p$$

Thus, the pairwise measure for differentiating class i from class j can be computed by:

$$M_{ij}(x_p) = \sum_{r=1}^{N_p} \min_{i,j} \left\{ P_{rp}^{(i)}, P_{rp}^{(j)} \right\}$$

The measure for differentiating class i from all other classes using x_p is defined by:

$$M_i(x_p) = \sum_{j \neq i}^{K} M_{ij}(x_p)$$

Finally, the overall measure of significance of x_p for differentiating all classes is computed as follows:

$$M(x_p) = \sum_{i=1}^{K} M_i(x_p) = \sum_{i=1}^{K} \sum_{j \neq i}^{K} M_{ij}(x_p)$$

The discriminant measure is the simplest measure and therefore is the fastest to compute. However, it can produce misleading results when the data classes are not unimodal. Consider, for example, the two marginal distributions shown in Figure 1- 16. The discriminant measure for X is quite small, since the separation between the class means relative to the sum of their variances is small; however, measurement X yields excellent between-class discrimination. This weakness is not a problem with the probability of confusion algorithm, since this latter is relatively independent of the functional form of the class distributions.

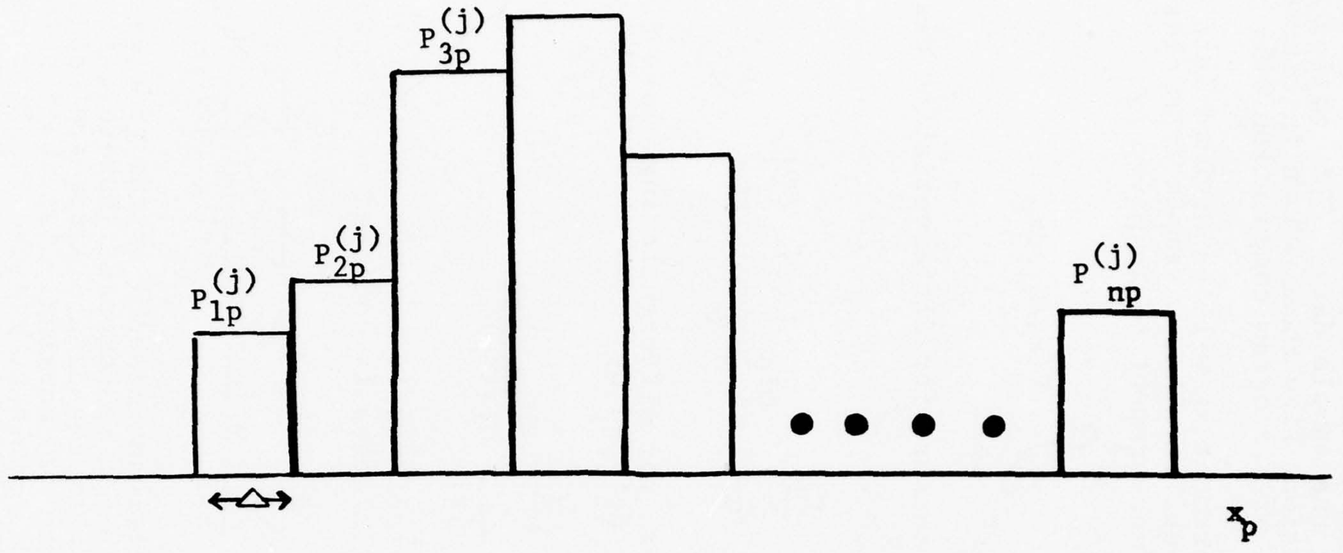


FIGURE 1 - 15

<u>Histogram Approximations of Marginal Class Distributions</u>

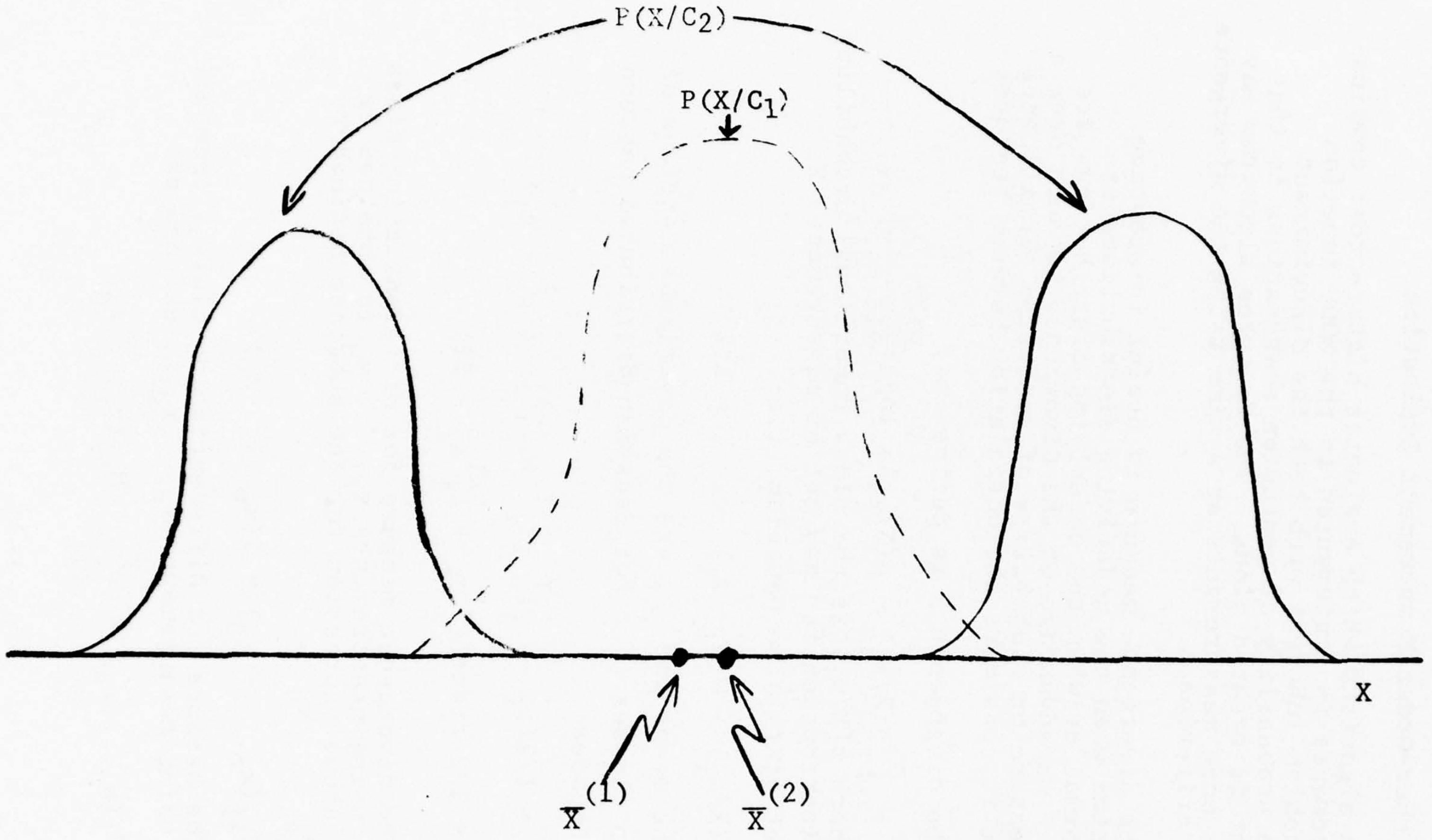


Figure 1-16: Example of Marginal Class Probability Distributions for which the Discriminant Measurement Algorithm Produces Misleading Results

1.3.1.3 Higher-Order Measurement Evaluation

An algorithm which evaluates higher-order combinations of measurements is implemented in the MOOS function features. A problem inherent with both the discriminant measure and the probability of confusion computations is that each measurement is treated alone. The features algorithm may evaluate one or more measurements at a time using the divergence measure as its criterion.

The divergence measure is useful in subspace feature evaluation when the underlying distributions are multivariate normal or when the underlying distributions are unimodal (4,5). An advantage of the divergence measure over a discriminant measure or probability of confusion significance measure is that it considers the correlation between features.

The divergence J is defined as:

$$J = \int [p(X/\omega_i) - p(X/\omega_j)] \log \left[\frac{p(X/\omega_i)}{p(X/\omega_j)}\right] dX$$

where $p[X/w_i]$ is the class conditional probability distribution for any set of measurements X.

Let $p(X/\omega_i)$ be Gaussian, i.e.,

$$p(X/\omega_i) \sim N(\mu_i', \Sigma_i) \qquad i = 1,2$$

where μ_i' are the means and Σ_i are the covariance matrices of the patterns in classes ω_i. For Gaussian-distributed pattern classes, this becomes:

$$J = 1/2(\mu_i' - \mu_j')^T [\Sigma_i^{-1} + \Sigma_j^{-1}] (\mu_i' - \mu_j')$$
$$+ 1/2 \operatorname{trace}(\Sigma_i^{-1}\Sigma_j + \Sigma_j^{-1}\Sigma_i - 2I)$$

The divergence measure for differentiating class i from class j using measurements $x_p, \ldots x_q$ is obtained by evaluating the above expression for the subspace defined by measurements $x_p, \ldots x_q$.

$$M_{ij}(x_p, \ldots x_q) = J(x_p, \ldots x_q)$$

The measure for differentiating class i from all other classes using measurements $x_p, \ldots x_q$ is defined as:

$$M_i(x_p, \ldots x_q) = \sum_{j \neq i}^{K} M_{ij}(x_p, \ldots x_q)$$

The measure for distinguishing all classes using measurements $x_p, \ldots x_q$ is defined as:

$$M(x_p, \ldots x_q) = \sum_{i=1}^{K} M_i(x_p, \ldots x_q) =$$

$$\sum_{i=1}^{K} \sum_{i \neq j}^{K} M_{ij}(x_p, \ldots x_q)$$

Since it is not practical to evaluate all possible combinations of the L measurements to determine an optimal feature subspace, a number of suboptimal search procedures are available.

The forward sequential suboptimal search procedure finds the "best" subset of N from the original L measurements using the measure for distinguishing all classes, $M(x_p, \ldots x_q)$. The first measurement selected is the best of the L measurements taken one at a time. The second measurement selected is the best of the L-1 remaining measurements when taken in combination with the first selected measurement. The third measurement selected is the best of the L-2 remaining measurements when taken in combination with the first and second selected measurement, and so on. The procedure halts when the user-specified value N is reached.

The union best by class approach to measurement selection utilizes the measure for distinguishing class i from all other classes, $M_i(x_p, \ldots x_q)$. The procedure is quite similar to the forward sequential technique except that at each step, more than one measurement may be selected. On the first round, the best measurement for distinguishing each of the K classes is selected, i.e., anywhere from 1 to K different measurements may be selected. The second round takes all remaining measurements in combination with the previously selected measurements, and again may add anywhere from 1 to K new measurements. The procedure halts when the user-specified value N is reached or exceeded.

The union best by class pair approach to measurement selection utilizes the measure for distinguishing class i from class j, $M_{ij}(x_p, \ldots x_q)$. The search algorithm is almost identical to the union best by class procedure. On the first round the best measurement for distinguishing each of the possible class pairs is selected, i.e., anywhere from 1 to K(K-1)/2 measurements may be selected. The second round (and all subsequent rounds) takes all the remaining measurements in combination with the previously selected measurements, and again may add anywhere from 1 to K(K-1)/2 new measurements. The procedure halts when the user-specified value N is reached or exceeded.

The measurement selection procedure has a great deal of flexibility in that the previously described techniques may be interactively mixed in any sequence. Furthermore, a preferred subset of the feature space may be selected as a starting point for the measurement selection computations.

1.3.2 Structure Analysis

The basic use of structure analysis in MOOS is in determining if the structure of the data for a particular class is unimodal or multimodal. If it is multimodal, it is frequently better to subdivide the class before attempting to design logic for distinguishing between classes. This is particularly true if the logic to be designed is statistically based.

All of the algorithms for structure analysis in MOOS involve projecting the data onto a one-or two-space and allowing the analyst to draw a partition(s) of the space if multimodality is present. All of the projections except one, NLM, are linear. The linear projections may also be used as the basis for group logic design.

No justification or explanation is given for coordinate or arbitrary vector projections.

1.3.2.1 The Eigenvector Plane (Least Squares)

The following section is an explanation and proof of the contention that planes defined by the two eigenvectors corresponding to the two largest eigenvalues of the estimated covariance matrix are optimal by the least squares criterion.

This can be shown as follows:

1. Define a plane by two unit orthogonal vectors $\underset{\sim}{e1}$ and $\underset{\sim}{e2}$ through a shifted origin denoted by $\underset{\sim}{d}$ ($\underset{\sim}{d}$ will turn out to be the mean of the data).

2. Set up an expression for the error (E) that arises in fitting the data by the plane described in (1). This will be obtained by summing the squared residuals from the plane.

3. Next, minimize the error E with respect to d, $\underset{\sim}{e1}$ and $\underset{\sim}{e2}$, under the constraints that $\underset{\sim}{e1}$ and $\underset{\sim}{e2}$ be unit vectors and orthogonal.

4. $\underline{d}$ will be found to be the mean vector. The eigenvectors of the estimated covariance matrix will be shown to be solutions to the minimization problem and, particularly, the two eigenvectors corresponding to the largest eigenvalues will turn out to be the desired solution. Note that there exists an infinite number of solutions, since any orthogonal rotation of $\underset{\sim}{e1}$, $\underset{\sim}{e2}$ in the plane defined by $\underset{\sim}{e1}$ and $\underset{\sim}{e2}$ will also be a solution; however, all these solutions describe the same least squares plane.

Let $\left.\begin{matrix}\underset{\sim}{X}_1\\ \vdots\\ \underset{\sim}{X}_N\end{matrix}\right\}$ be L-dimensional data vectors

$\underset{\sim}{d}$ = new origin for the data

$\underset{\sim}{e}_1$, $\underset{\sim}{e}_2$ define the plane through the new origin.

Define $\underset{\sim}{Y}_i = \underset{\sim}{X}_i - \underset{\sim}{d}$

The residual distance squared from the fitting plane for the Kth data vector is given by

$$\left|\underset{\sim}{r}_k\right|^2 = \underset{\sim}{r}_k^t \underset{\sim}{r}_k = \left|\underset{\sim}{Y}_k - (\underset{\sim}{Y}_k^t \underset{\sim}{e}_1)\underset{\sim}{e}_1 - (\underset{\sim}{Y}_k^t \underset{\sim}{e}_2)\underset{\sim}{e}_2\right|^2$$

The fitting error is given by the summation of the squared residual, i.e.

$$E = \sum_{k=1}^{N} r_k^t r_k = \sum_{k=1}^{N}\left\{\underset{\sim}{Y}_k^t \underset{\sim}{Y}_k - (\underset{\sim}{Y}_k^t \underset{\sim}{e}_1)^2 - (\underset{\sim}{Y}_k^t \underset{\sim}{e}_2)^2\right\}$$

Using Lagrange multipliers to account for the constraints on $\underset{\sim}{e}_1$ and $\underset{\sim}{e}_2$, we obtain

$$E^* = \sum_{k=1}^{N}\left\{\underset{\sim}{X}_k^t \underset{\sim}{X}_k - 2\underset{\sim}{X}_k^t \underset{\sim}{d} + \underset{\sim}{d}^t \underset{\sim}{d} - (\underset{\sim}{X}_k^t \underset{\sim}{e}_1 - \underset{\sim}{d}^t \underset{\sim}{e}_1)^2 - (\underset{\sim}{X}_k^t \underset{\sim}{e}_2 - \underset{\sim}{d}^t \underset{\sim}{e}_2)^2\right\}$$

$$-\lambda_1\left[\underset{\sim}{e}_1^t \underset{\sim}{e}_1 - 1\right] - \lambda_2\left[\underset{\sim}{e}_2^t \underset{\sim}{e}_2 - 1\right] - \lambda_3 \underset{\sim}{e}_1^t \underset{\sim}{e}_2$$

Taking the partial with respect to $\underset{\sim}{d}$ we obtain

$$0 = \frac{\partial E^*}{\partial \underset{\sim}{d}} = \sum_{k=1}^{N}\left\{-2\underset{\sim}{X}_k + 2\underset{\sim}{d} + 2(\underset{\sim}{X}_k^t \underset{\sim}{e}_1 - \underset{\sim}{e}_1^t \underset{\sim}{d})\underset{\sim}{e}_1 + 2(\underset{\sim}{X}_k^t \underset{\sim}{e}_2 - \underset{\sim}{e}_2^t \underset{\sim}{d})\underset{\sim}{e}_2\right\}$$

Let $\underset{\sim}{U} = \frac{1}{N}\sum_{k=1}^{N} \underset{\sim}{X}_k$

Substitute

$$\frac{\partial E^*}{\partial \underset{\sim}{d}} = 0 = 2N \left[(\underset{\sim}{U}-\underset{\sim}{d}) + \left\{ (\underset{\sim}{U}-\underset{\sim}{d})^t \underset{\sim}{e}_1 \right\} \underset{\sim}{e}_1 + \left\{ (\underset{\sim}{U}-\underset{\sim}{d})^t \underset{\sim}{e}_2 \right\} \underset{\sim}{e}_2 \right]$$

$\therefore$ $\underset{\sim}{U}=\underset{\sim}{d}$ is a solution. Thus $\underset{\sim}{d}$ is the data mean.

continuing

$$\frac{\partial E^*}{\partial \underset{\sim}{e}_1} = \left\{ 2 \sum_{k=1}^{N} \left[\underset{\sim}{Y}_k \underset{\sim}{Y}_k^t \right] \underset{\sim}{e}_1 \right\} - 2\lambda_1 \underset{\sim}{e}_1 - \lambda_3 \underset{\sim}{e}_2 = 0$$

$$\frac{\partial E^*}{\partial \underset{\sim}{e}_2} = \left\{ 2 \sum_{k=1}^{N} \left[\underset{\sim}{Y}_k \underset{\sim}{Y}_k^t \right] \underset{\sim}{e}_2 \right\} - 2\lambda_2 \underset{\sim}{e}_1 - \lambda_3 \underset{\sim}{e}_1 = 0$$

Note that the estimated covariance matrix Σ, is given by

$$\Sigma = \frac{1}{n-1} \sum_{k=1}^{N} \left[\underset{\sim}{Y}_k \underset{\sim}{Y}_k^t \right]$$

Let $\hat{\Sigma} = (N-1) \Sigma$, and substitute above

$$2 \hat{\Sigma} \underset{\sim}{e}_1 - 2\lambda_1 \underset{\sim}{e}_1 - \lambda_3 \underset{\sim}{e}_2 = 0 \qquad \text{(A)}$$

$$2 \hat{\Sigma} \underset{\sim}{e}_2 - 2\lambda_2 \underset{\sim}{e}_2 - \lambda_3 \underset{\sim}{e}_1 = 0 \qquad \text{(B)}$$

Multiply (A) from the left by $\underset{\sim}{e}_1^t$

Multiply (B) from the left by $\underset{\sim}{e}_2^t$

Multiply (A) from the left by $\underset{\sim}{e}_2^t$
to obtain

$$\underset{\sim}{e}_1^t \hat{\Sigma} \underset{\sim}{e}_1 = \lambda_1 \qquad (1)$$

$$\underset{\sim}{e}_2^t \hat{\Sigma} \underset{\sim}{e}_2 = \lambda_2 \qquad (2)$$

$$2\underset{\sim}{e}_2^t \hat{\Sigma} \underset{\sim}{e}_1 = \lambda_3 \qquad (3)$$

Substitute back into (A) and (B)

$$\text{(A)} \quad 2\hat{\Sigma}\,\underset{\sim}{e}_1 - 2\left[\underset{\sim}{e}_1^t\,\hat{\Sigma}\,e_1\right]\underset{\sim}{e}_1 \quad -2 \quad \left[\underset{\sim}{e}_2^t\,\hat{\Sigma}\,\underset{\sim}{e}_1\right]\underset{\sim}{e}_2 = 0$$

$$\text{(B)} \quad 2\hat{\Sigma}\,\underset{\sim}{e}_2 - 2\left[\underset{\sim}{e}_2^t\,\hat{\Sigma}\,\underset{\sim}{e}_2\right]\underset{\sim}{e}_2 \quad -2 \quad \left[\underset{\sim}{e}_2^t\,\hat{\Sigma}\,\underset{\sim}{e}_1\right]\underset{\sim}{e}_1 = 0$$

Now let $\hat{\Sigma}\,\underset{\sim}{e}_1 = \alpha_1\,\underset{\sim}{e}_1$ and

$\hat{\Sigma}\,\underset{\sim}{e}_2 = \alpha_2\,\underset{\sim}{e}_2$ and substitute

into (A) and (B)

$$\text{(A)} \Rightarrow 2\,\alpha_1 \underset{\sim}{e}_1 - 2\,\alpha_1 \underset{\sim}{e}_1 = 0$$

$$\text{(B)} \Rightarrow 2\,\alpha_2 \underset{\sim}{e}_2 - 2\,\alpha_2 \underset{\sim}{e}_2 = 0$$

$\therefore$ A solution plane is given by the two eigenvectors of the estimated covariance matrix $\hat{\Sigma}$.

The least squared error is given by

$$E = R - \underset{\sim}{e}_1^t\,\hat{\Sigma}\,\underset{\sim}{e}_1 - \underset{\sim}{e}_2^t\,\hat{\Sigma}\,\underset{\sim}{e}_2$$

where $R = \text{constant} = \sum_{k=1}^{n} \underset{\sim}{Y}_k^t\,\underset{\sim}{Y}_k$

or equivalently

$$E = R - \alpha_1 - \alpha_2$$

Therefore, the error is minimized by selecting the two eigenvectors corresponding to the two largest eigenvalues.

It can similarly be shown that the projection on the eigenvector associated with the largest eigenvalue is the best (by the least squares criterion) one-space projection.

1.3.2.2 Discriminant Projections

The MOOS functions ardg\$_ _ _ and asdg\$_ _ _ offer the analyst another projection direction or plane. The only difference between the two functions is in how the two classes upon which the projection is based are determined. The two classes may be composed of any two classes of the current data set or they may be composed of any two groups of classes which are "lumped together" for the purpose of determining the projection direction(s).

The entire current data set is projected into the space defined by the Fisher Discriminant $\underset{\sim}{d}_1$ and a second vector $\underset{\sim}{d}_2$, where $\underset{\sim}{d}_2$ is that direction which maximizes the projected between-class scatter relative to the sum of the projected within-class scatter, under the constraint that $\underset{\sim}{d}_2$ be orthogonal to $\underset{\sim}{d}_1$. In summary,

$$\underset{\sim}{d}_1 = \alpha_1 W^{-1} \underset{\sim}{\Delta}$$

$$\underset{\sim}{d}_2 = \alpha_2 \left[W^{-1} - \left(\underset{\sim}{\Delta}^T \left[W^{-1}\right]^2 \underset{\sim}{\Delta} / \underset{\sim}{\Delta}^T \left[W^{-1}\right]^3 \underset{\sim}{\Delta} \right) \left[W^{-1}\right]^2 \right] \underset{\sim}{\Delta}$$

where α_1 and α_2 are normalizing constants

$\underset{\sim}{\Delta}$ = the difference between the class mean vectors, $\underset{\sim}{\mu}_1 - \underset{\sim}{\mu}_2$

W = sum of the within-class scatter matrices

Notice that both $\underset{\sim}{d}_1$ and $\underset{\sim}{d}_2$ are computed using W^{-1}. If the data lies in a subspace, then it can be shown that W will be singular. If the data is approximately contained in any subspace, then W will at best be ill-conditioned. In either case, the numerical computation of W^{-1} will be extremely tenuous. Thus, prior to computing W^{-1}, W must first be checked to determine if it is ill-conditioned or singular. In either case, we will compute a subspace such that when the data is orthogonally projected onto this subspace, the $W_{new} = TW_{old}T^T$ will be well-conditioned. Next, $\underset{\sim}{d}_1$ and $\underset{\sim}{d}_2$ will be computed in the subspace using W_{new}, and finally, we will transform $\underset{\sim}{d}_1$ and $\underset{\sim}{d}_2$ back to the original L-dimensional space.

If the one-space option is chosen the data is projected on $\underset{\sim}{d}_1$, that is, in the Fisher direction only.

1.3.2.3 Generalized Discriminant Projections

The MOOS function gndv$_ _ _ offers the analyst the capability of projecting data onto a discriminant direction or plane which has been optimized to produce maximum discrimination for all classes. This is a generalization of the Fisher discriminant projection described in Section 1.3.2.2.

The Fisher discriminant is obtained by solving for the unit vector d which maximizes the following ratio:

$$R = \frac{d^T B d}{d^T W d}$$

where B is the between-class scatter matrix, and W is the sum of the within-class scatter matrices.

To solve for the generalized Fisher discriminant directions, we take the vector derivative of the above ratio R with respect to d and set the resultant equation to zero. The procedure generates the following generalized eigenvector equation.

$$\left[B - \lambda W \right] d = 0$$

$$\left[W^{-1} B - \lambda I \right] d = 0$$

The generalized discriminant vectors are the eigenvectors of the non-symmetric matrix $W^{-1}B$. The rank of the between-class scatter matrix for the K-class discrimination problem is K-1, therefore, no more than K-1 nonzero eigenvector solutions exist. Thus, the generalized discriminant vector function produces K-1 discriminant vectors, with the vectors which correspond to the largest eigenvalues producing the maximum discrimination. The Gram-Schmidt orthonormalization technique is applied to the eigenvectors to insure that they are orthogonal unit vectors (6).

1.3.2.4 Nonlinear Mapping

The Nonlinear Mapping Algorithm (NLM) is based upon a point mapping of the N L-dimensional vectors from the L-space to a lower-dimensional space such that the inherent structure of the data is approximately preserved under the mapping. The approximate structure preservation is accomplished by fitting N points in the lower-dimensional space such that their interpoint distances approximate the corresponding interpoint distances in the L-space.

Suppose that we have N vectors in an L-space designated X_i, i = 1, ..., N and, corresponding to these, we define N vectors in the two-space designated Y_i, i = 1, ..., N. Let the distance between the vectors X_i and X_j in the L-space be defined by

$$d_{ij}^* = \text{dist}\left[X_i, X_j\right]$$

and the distance between the corresponding vectors Y_i and Y_j in the two-space be defined by

$$d_{ij} = \text{dist}\left[Y_i, Y_j\right]$$

Let us now randomly[1] choose an initial two-space configuration for the Y vectors and denote this configuration as follows:

$$Y_1 = \begin{bmatrix} y_{11} \\ y_{12} \end{bmatrix} \qquad Y_2 = \begin{bmatrix} y_{21} \\ y_{22} \end{bmatrix} \; . \; . \; . \quad Y_N = \begin{bmatrix} y_{N1} \\ y_{N2} \end{bmatrix}$$

1 - For the purpose of this discussion it is convenient to think of the starting configuration as being selected randomly; however, in practice the initial configuration for the vectors is found by projecting the L-dimensional data orthogonally onto a two-space spanned by the two original coordinates with largest variances.

Next we compute all the two-space interpoint distances d_{ij}, which are then used to define an error E; E is a measure of how well the present configuration of N points in the two-space fits the N points in the L-space, i.e.,

$$E = \frac{1}{\sum_{i<j} \left[d_{ij}^*\right]} \sum_{i<j}^{N} \frac{\left[d_{ij}^* - d_{ij}\right]^2}{d_{ij}^*} .$$

Note that the error is a function of the 2*N variables y_{pq}, $p = 1, \ldots, N$ and $q = 1, 2$. The next step in the NLM algorithm is to adjust the y_{pq} variables or, equivalently, change the 2-space configuration so as to decrease the error. We use a steepest descent procedure to search for a minimum error.

Let E(m) be defined as the mapping error after the mth iteration, i.e..

$$E(m) = \frac{1}{c} \sum_{i<j}^{N} \left[d_{ij}^* - d_{ij}(m)\right]^2 / d_{ij}^*$$

where

$$c = \sum_{i<j}^{N} \left[d_{ij}^*\right]$$

and

$$d_{ij}(m) = \sqrt{\sum_{k=1}^{2} \left[y_{ik}(m) - y_{jk}(m)\right]^2} .$$

The new 2-space configuration at time m + 1 is given by

$$y_{pq}(m+1) = y_{pq}(m) - (MF) \cdot \Delta_{pq}(m)$$

where

$$\Delta_{pq}(m) = \frac{\partial E(m)}{\partial y_{pq}(m)} \Bigg/ \left| \frac{\partial^2 E(m)}{\partial y_{pq}(m)^2} \right|$$

and MF is the "magic factor" which was determined empirically to be MF $\approx$ 0.3. The partial derivatives are given by

$$\frac{\partial E}{\partial y_{pq}} = \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq p}}^{N} \left[\frac{d^*_{pj} - d_{pj}}{d_{pj}\ d^*_{pj}} \right] (y_{pq} - y_{jq})$$

and

$$\frac{\partial^2 E}{\partial y_{pq}^2} = \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq p}}^{N} \frac{1}{d^*_{pj} \cdot d_{pj}} \left[(d^*_{pj} - d_{pj}) - \frac{(y_{pq} - y_{jq})^2}{d_{pj}} \left(1 + \frac{d^*_{pj} - d_{pj}}{d_{pj}} \right) \right]$$

In our program we take precautions to prevent any two points in the two-space from becoming identical. This prevents the partials from "blowing up."

Because the number of computations required in the NLM algorithm is approximately proportional to $N^2/2$ (where N is the number of vectors), the MOOS implementation of NLM has an upper limit of 200 vectors. If the number of vectors in the current data set exceeds 200, a reduction is required; the algorithm for performing this reduction is explained below.

The user specifies the number of vectors (or cluster centers) to which he wishes the set to be reduced. If the current data set contains more than one class, the user also specifies whether he wants the number of vectors in each class of the new set to be the same, or proportional to the number of vectors in each class of the original data set. The reduction process is then performed on one class at a time.

Given that M is the number of vectors in the original class, and N ($N < M$) is the number of vectors desired for the reduced class, then the number of original vectors lumped together to produce a reduced vector is $M/N=K$. Each of the N reduced vectors is thus the mean vector of K vectors from the original class. The selection as to which K vectors are to be clustered together is made as follows: The entire set of vectors is searched; the vector which is farthest away (in the Euclidean sense) from the mean of the entire class is picked as a starting vector. Then the K-1 closest vectors to that starting vector are found, and the mean vector of those $1+(K-1)=K$ vectors is taken as the reduced vector or cluster center. The starting vector for the next cluster center is found from among the remaining vectors by searching for the one furthest away from the previous cluster; the clustering process is repeated as often as necessary. (If K is not an integer, then the first A ($A < N$ where A is the remainder of M/N) cluster centers will be the mean of $[K] + 1$ vectors and the remaining N-A cluster centers will be the mean of $[K]$ vectors.)

1.3.3 Logic Design

In general, the primary goal of a pattern classification analyst is to design a logic, or series of tests, which will, with a suitable degree of accuracy, assign an unlabeled vector from the feature space to a particular class (or reject it as unclassifiable within the required degree of probability).

MOOS provides the analyst with several types of logic design algorithms and variations within each type.

1.3.3.1 Group Logic Design

In group logic, the analyst makes an interactive, subjective decision and actually participates in the logic design process. The particular node of the logic tree for which logic is being designed is examined; the vectors from the classes present there are projected on a one- or two-space. If there is (in the analyst's judgment) sufficient separation between classes, or between groups of classes, he may draw one or two boundaries so that the feature space is partitioned into two or three regions. These regions are then labeled as to the class or classes present in them (a region may be labeled as the null class or reject region). This is illustrated in Figure 1-17.

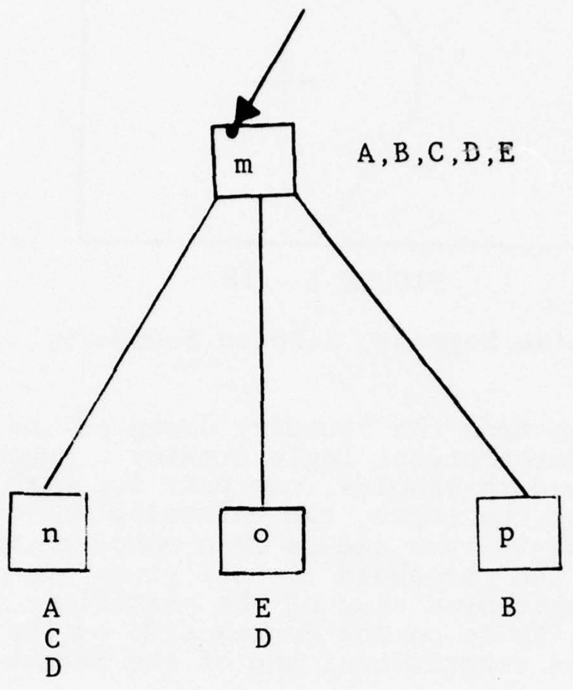


FIGURE 1 - 17: LOGIC TREE NODE - GROUP LOGIC

In this example, group logic which was designed at node m separated the five classes present (A,B,C,D, and E) into three groups. Class B was completely separable from the other classes and was assigned to node p of the logic tree; the remaining portions of the feature space, assigned to nodes n and o, contain the groups of classes A,C and D, and E and D, respectively. Notice that the samples from class D fell into both regions; this is permissible.

For the one-space implementation of these logics, the mathematics is extremely simple. The unlabeled vector to be classified is merely projected (dot product) onto the projection direction (discriminant); the value of this scalar is then compared to the value of the boundary (threshold) drawn by the user.

Two-space logic mathematics is slightly more complicated; it is illustrated below.

When the user defines a two-space boundary, he draws, on the projection plane, from one to five connected line segments which must define a convex region; he then draws a reference point indicating which is the convex side. See Figure 1-18 on the following page.

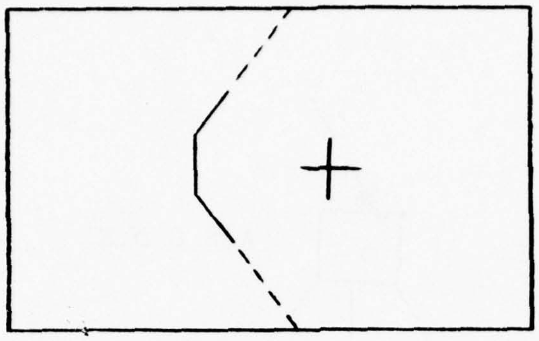


FIGURE 1 - 18

3-Line Segment, 2-Space Boundary.

The transition from the boundary drawn on the two-space projection to the mathematical logic creates a sequence of discriminant vectors and thresholds, one pair for each line segment. In the evaluation of the logic, the unlabeled vector is projected on each discriminant in turn and is then compared to the threshold. If it is less than the threshold for any given line segment, the vector is on the non-convex side of the partition; if it is greater than the threshold, it is on the convex side of the partition. The determination of the discriminant and of the threshold for a line segment follows.

Given three points on the projected plane

point S (boundary start point)
point E (boundary end point)
point C (point on "convex" side of boundary)

if these are considered as vectors (in the projected two-space), i.e.

$$\underset{\sim}{S} = \langle x_S, y_S \rangle$$

$$\underset{\sim}{E} = \langle x_E, y_E \rangle$$

$$\underset{\sim}{C} = \langle x_C, y_C \rangle$$

then a vector $\underset{\sim}{D}$ normal to the boundary line in the projected two-space is given by: $\langle d_1, d_2 \rangle$ where $d_1 = +(y_E - y_S)$ and $d_2 = -(x_E - x_S)$.

Project the convex point and boundary end point onto this vector.

Let $$P_C = \underset{\sim}{C} \cdot \underset{\sim}{D}$$

$$P_E = \underset{\sim}{E} \cdot \underset{\sim}{D}$$

Then if $P_C \geq P_E$ save P_E as the discriminant threshold and compute and store the discriminant vector.

or if $P_C < P_E$ replace $\underset{\sim}{D}$ by $\langle -d_1, -d_2 \rangle$, save $-P_E$ as the discriminant threshold and compute the discriminant vector.

The discriminant vector = $d_1 \underset{\sim}{X} + d_2 \underset{\sim}{Y}$, where $\underset{\sim}{X}$ and $\underset{\sim}{Y}$ are the L-dimensional projection vectors used to project a point in L-space onto the projection plane (on which the boundary was drawn) for the purpose of evaluating logic (i.e. deciding if any point V in L-space lies on the "convex" side of the L-space hyperplane determined by the boundary $(\underset{\sim}{E}-\underset{\sim}{S})$ drawn on the projection plane).

If $\underset{\sim}{V} \cdot \underset{\sim}{A} \geq$ threshold

where $\underset{\sim}{V}$ = L-dimensional vector for point V, and

$\underset{\sim}{A}$ = the discriminant vector,

then the point V is on the convex side of the boundary.

1.3.3.2 Complete Within-Group Logic

This type of logic creates a node, or region within the feature space, for each individual class present at the node for which logic is being designed. The logic tree representation of this is illustrated in Figure 1-19.

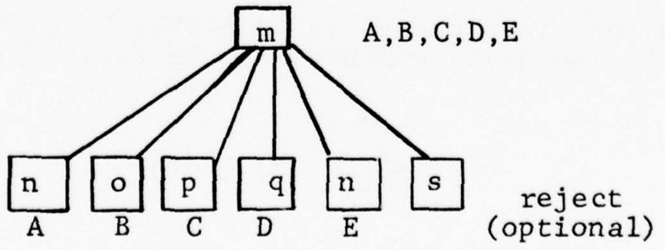


FIGURE 1-19
Complete Within-Group Logic Tree Node (5 classes)

This type of logic is more statistically based and requires less user interaction than group logic, where the analyst must determine the boundary(s) himself. MOOS does, however, allow modifications of these logics wherein the analyst may make a considerable number of subjective decisions. The three variations of complete within-group logic are Nearest Mean Vector (NMV) logic, pairwise logic, and closed decision boundary logic.

1.3.3.2.1 NMV Logic

Generalized NMV logic is a k-class classification technique; it classifies an unknown vector from the feature space according to a metric, which is computed from the unknown vector to the mean vectors of the k classes of the design set. The decision is in favor of the class which produces the minimum value of the metric. The generalized metric is:

$$d_i = \sqrt{(\underset{\sim}{X} - \underset{\sim}{M_i})^T C_i^{-1} (\underset{\sim}{X} - \underset{\sim}{M_i})}$$

where $\underset{\sim}{X}$ = the ℓ-dimensional unknown feature vector

$\underset{\sim}{M_i}$ = the ℓ-dimensional mean vector for class i

C_i = an $\ell \times \ell$ matrix

If C_i is the covariance matrix for class i of the design set, then the metric is known as the Mahalanobis distance.

If C_i is the identity matrix, the metric is simply the Euclidean distance from the unknown vector to the mean vector of each class.

In MOOS, three basic options of NMV Logic are available; a reject strategy can be specified under each option. To reduce unnecessary calculation we use the square of the metric.

In the first option (simple NMV) C is the identity matrix, and the metric is computed in the form:

$$d_i = \sum_{j=1}^{\ell} (x_j - \mu_j^i)^2$$

where $\underset{\sim}{X}$ = unknown vector = $(x_1, x_2, \ldots, x_\ell)$

$\underset{\sim}{M_i}$ = mean vector of class i=$(\mu_1^i, \mu_2^i, \ldots, \mu_\ell^i)$

In the second option (weighting vectors), C_i is a diagonal matrix whose elements are the variances of the ℓ components of the design set samples of class i. The computational form of the metric used in this option is:

$$d_i = \sum_{j=1}^{\ell} \frac{(x_j - \mu_j^i)^2}{V_j^i}$$

where V_j^i = variance of jth component of the ith class

In the third option (weighting matrix), C_i is the covariance matrix of the design set samples of class i. The computation in this case involves the actual vector times matrix times vector multiplication, as defined by the generalized metric formula.

The optional reject strategy allows the user to specify a reject distance. In this case the decision strategy is: decide on the class j for which d_j is the minimum of all d_i, i = 1, ..., k, if and only if d_j is less than the reject distance, otherwise reject. The user can specify a separate reject distance for each class, or he may use the same reject distance for all classes. This strategy may be used with any of the three metrics.

1.3.3.2.2 Pairwise Logic

In MOOS, pairwise logic is created by the routine fisher. This routine creates a one-space logic based on the Fisher direction (see section 1.3.2.2) for each possible pair of classes from among the classes present at the node for which the logic is being designed. Given N classes at the node, this will produce N(N-1)/2 class pairs. Each of these class-pair logics classifies (or can be thought of as producing a vote for) a vector as one or the other of these classes (or reject, depending upon the number of thresholds selected - see section 1.2).

In the evaluation of pairwise logic, the unlabeled vector is evaluated by each of the N(N-1)/2 class-pair logics and a "vote count" is kept for each class. After all class pairs have been evaluated, the vector is classified according to the vote counts. It is classified as belonging to that class which received the maximum vote count, provided this maximum is greater than or equal to a user-specified vote count threshold. In case of a tie for the maximum vote count, an attempt is made to break the tie by referring to the a priori class probabilities; if these are also equal, the vector is rejected.

The flow of pairwise logic evaluation is illustrated in Figure 1-21.

MOOS also allows the user, through the routine pairmod, the capability of modifying each of the class-pair logics. The allowable types of logic are:

1) Fisher (1 to 5 thresholds)
2) Any arbitrary one-space projection vector
3) Optimal discriminant plane
4) Any arbitrary two-space plane
5) Boolean

1.3.3.2.3 Closed Decision Boundary Logic

A closed decision boundary logic strategy is implemented in the routine closedcn. This program creates an L-dimensional hyperregion to enclose each of the classes in the selected data set. Three types of hyperregion are available: hyperrectangular, hyperspherical, and hyperellipsoid.

The evaluation of hyperrectangular logic is performed as follows:

1) Project the unknown sample vector on the basis vectors.

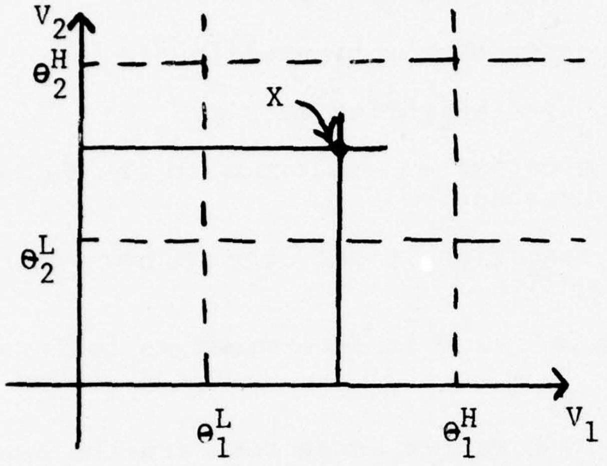
$Y_j = \underset{\sim}{X} \cdot \underset{\sim}{V}_j$

$\underset{\sim}{X}$ = the unknown vector

Y_j = the jth component of the projected vector

$\underset{\sim}{V_j}$ = the jth basis vector

2) The unknown vector is tested against a high and a low threshold along each basis vector. The vector is in the hyperrectangle if and only if its projection on each of the basis vectors is within the high and low thresholds for each basis vector. A two-dimensional case is illustrated in Figure 1-20.



θ_i^L = low threshold on basis vector V_i

θ_i^H = high threshold on basis vector V_i

The point X lies between both pairs of thresholds on the basis vectors and is therefore inside the hyperrectangle.

Figure 1-20

The evaluation of hyperspherical closed decision boundary logic is performed by calculating the Euclidean distance between the unknown vector and the center vector of the hypersphere. If this distance is less than or equal to the radius of the hypersphere, then the unknown vector is inside the hypersphere.

$$d^2 = \sum_{i=1}^{L} (X_i - M_i)^2$$

M_i = i th component of the center vector of the hypersphere.

X_i = i th component of the unknown vector.

d = Euclidean distance between $\underset{\sim}{X}$ and $\underset{\sim}{M}$ (d^2 is used rather than d to reduce computation).

The evaluation of hyperellipsoid closed decision boundary logic is performed by doing the following calculation for an unknown vector.

$$(\underset{\sim}{X} - \underset{\sim}{M})^T W (\underset{\sim}{X} - \underset{\sim}{M}) \leq C$$

$\underset{\sim}{X}$ = the unknown vector

$\underset{\sim}{M}$ = the center of the hyperellipsoid

W = an L-by-L weighting matrix

C = a size parameter analogous to the radius of a hypersphere.

If the above condition is met, the unknown vector lies inside the hyperellipsoid.

The weighting matrix W is determined as follows:

$$W = \left[B^T A B\right]^{-1}$$

B = an L-by-L matrix whose rows are the axis vectors of the hyperellipsoid (the only axis vectors currently implemented are the eigenvectors of the covariance matrix of the class)

A = an L-by-L diagonal matrix. A_{ii} = the length of the i^{th} axis of the hyperellipsoid

In the case where A is a diagonal matrix of eigenvalues and the center vector $\underset{\sim}{M}$ is the mean of a class: W is the inverse covariance matrix and C is the Mahalanobis distance.

Each class of a given data set may have any one of the three types of hyperregion surrounding it. Three cases arise depending on how many hyperregions an unknown vector falls into. If an unknown vector does not lie in any hyperregion, it is rejected. If an unknown vector falls in one hyperregion, it is assigned to the class associated with that hyperregion. If an unknown vector lies in more than one hyperregion (referred to as overlap in further discussion), it is rejected unless the user has specified otherwise.

"Overlap" vectors may be placed in a new data tree, and further classification logic developed on the new data tree to reduce the number of rejections produced by closed decision boundary logic. This is a non-standard approach in that a data set must be passed against two independent logic trees to produce the final classification results.

When an unknown vector is rejected due to an overlap condition, some useful information may be retained. If the vector fell into only a few of the possible hyperregions, at least the number of choices as to which class the vector really belongs has been narrowed. This partial classification information may be utilized by Fisher pairwise logic.

The evaluation of Fisher pairwise logic performed on a data tree consisting of "overlap" vectors differs from the usual pairwise evaluation. Each unknown vector is tested only by pairwise decisions involving the possible classes indicated by closed decision boundary logic. Partial classification information obtained from a closed decision boundary evaluation may be utilized only by pairwise logic.

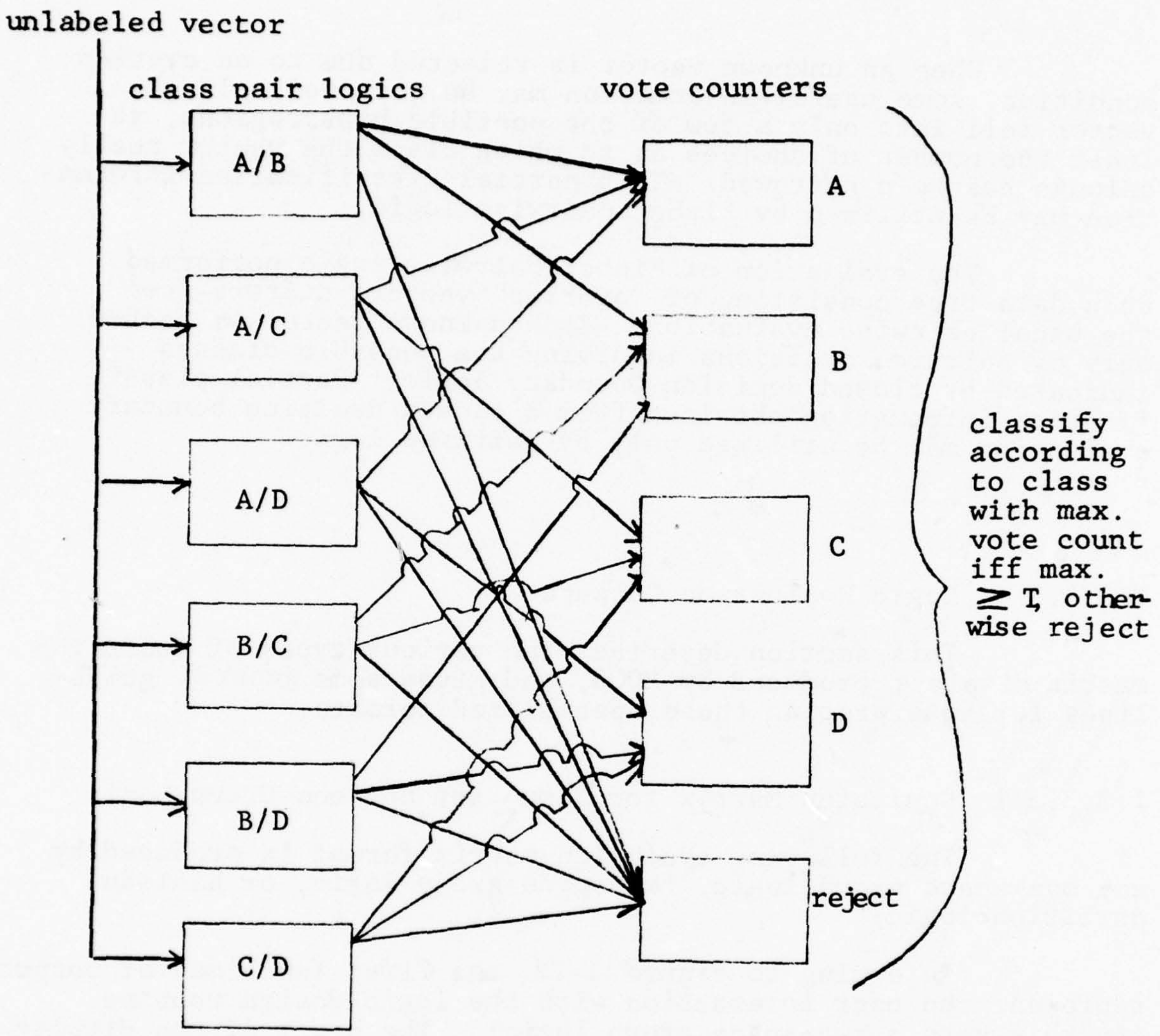
1.3.3.3 Logic Evaluation Outputs

This section describes the various types of confusion matrix displays produced by MOOS, and gives some general guidelines for interpreting these specialized formats.

1.3.3.3.1 Confusion Matrix for Temporary Between-Group Logic.

The following confusion matrix format is produced by any one-space group logic, two-space group logic, or Boolean partition logic.

Referring to Figure 1-22, the first few lines of output represent the user interaction with the logic design routine (in this case a two-space group logic). The heart of the display consists of a matrix format in which the columns are associated with nodes in the logic tree structure, and the rows correspond to the data classes in the data set being evaluated. Any particular element of the matrix is the number of vectors from a given class which were assigned to a particular logic node. The leftmost column of numbers always refers to the node on which logic was designed. To the right of and below the matrix are various totals and percentages designed to aid the analyst in the interpretation of these results.


unlabeled vector
class pair logics
vote counters
A/B
A/C
A/D
B/C
B/D
C/D
A
B
C
D
reject
classify according to class with max. vote count iff max. ≥ T, otherwise reject

The output of each class-pair logic can be one of the following:

a vote for no class (reject)
or
a vote for the first class of the pair
or
a vote for the second class of the pair

FIGURE 1-21

PAIRWISE LOGIC (4 CLASSES)

```
Enter the display symbols of the classes on the convex side of boundary 1
(on one line - no delimiters)
cosw
the logic node assigned to these dataclasses is 2
Enter the display symbols of the classes on the excess side of the boundary(s)
avco
the logic node assigned to these classes is 3

          logic node
         1     2     3      cor  %cor     err   %err
soys    61    61     0       61 100.0       0    0.0
corc    60    23    37       60 100.0       0    0.0
oato    65    53    12       65 100.0       0    0.0
whew    60    60     0       60 100.0       0    0.0
clov    62     0    62       62 100.0       0    0.0
alfa    54     0    54       54 100.0       0    0.0
ryer    60    60     0       60 100.0       0    0.0

tot    422   257   165      422 100.0       0    0.0
cor    422   257   165
%cor 100.0 100.0 100.0
err      0     0     0
%err   0.0   0.0   0.0
Do you want a hardcopy with a listing of misclassified vectors?
```

Figure 1-22: Confusion Matrix for Tempcrary Between-Group Logic

If the user decides to produce a high-speed printer copy of this confusion matrix, he may choose to get a listing of all incorrectly classified vectors. Each error is listed by data class, vector identification number, and the logic node to which it was assigned.

1.3.3.3.2 Confusion Matrix for Temporary Within-Group Logic

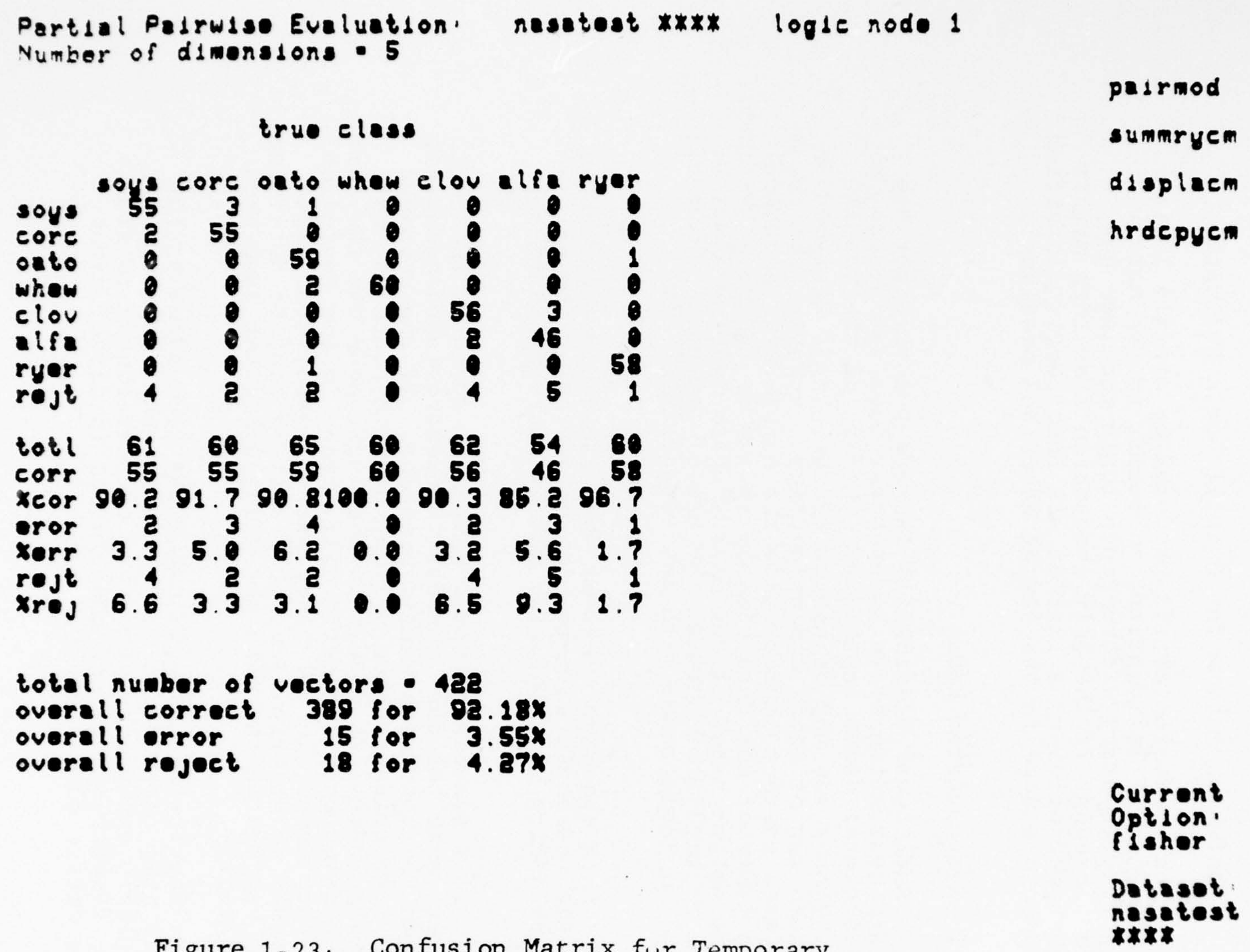
The following confusion matrix format is produced by the within-group logic routines nmv, fisher, and closedcn. Referring to Figure 1-23, the first two lines of output describe the type of within-group logic being evaluated, the data set on which logic was designed, and the number of dimensions. The heart of the display consists of a matrix format in which the column labels correspond to the data classes of the data set being evaluated, and the row labels are associated with the classes in the data set on which logic was designed. Any particular element of this matrix is the number of vectors from a given data class which were assigned to a particular logic node. (In the case where all classification was correct, all off-diagonal elements of the matrix would be zero.) Below this matrix are various totals and percentages designed to aid the analyst in the interpretation of his results.

If the user decides to produce a high-speed printer copy of a confusion matrix, he may choose to get a listing of all incorrectly classified vectors. For both nmv, fisher, and closedcn, each error is listed by data class, vector identification number and the logic node to which it was assigned. The following additional useful information is listed for each vector:

In the case of nmv, for each misclassified vector, the distance to the true class and the distance to the assigned class is listed. If the vector was rejected, the distance to the closest class is listed rather than the distance to the assigned class.

In the case of fisher, the first additional line usually begins with the phrase "lost to:" followed by a list of display symbols. Each display symbol refers to an incorrect pairwise decision involving the true class. If the vector was assigned to the wrong class by a pairwise decision box, the display symbol of the incorrect class is listed. An "r" in parentheses immediately following a class symbol indicates that the vector was rejected, not misclassified, by that decision box. The second line contains a list of vote counts for the given vector, in order of ascending logic node number. The last vote count listed is always the value of the reject vote count. If there was a tie situation, the first additional line of output is preceded by "tie" or "favorably broken tie" (see Section 1.3.3.2.2).

Closed decision boundary logic (closedcn) lists the type of hyperregion associated with the true class. If a vector falls into more than one hyperregion, the names of the classes associated with those hyperregions are also listed.



```
Partial Pairwise Evaluation·   nasatest ****   logic node 1
Number of dimensions = 5
                                                                  pairmod
              true class
                                                                  summrycm
       soys corc oato whew clov alfa ryer
soys    55    3    1    0    0    0    0                           displacm
corc     2   55    0    0    0    0    0
oato     0    0   59    0    0    0    1                           hrdcpycm
whew     0    0    2   60    0    0    0
clov     0    0    0    0   56    3    0
alfa     0    0    0    0    2   46    0
ryer     0    0    1    0    0    0   58
rejt     4    2    2    0    4    5    1

totl    61   60   65   60   62   54   60
corr    55   55   59   60   56   46   58
%cor  90.2 91.7 90.8100.0 90.3 85.2 96.7
eror     2    3    4    0    2    3    1
%err   3.3  5.0  6.2  0.0  3.2  5.6  1.7
rejt     4    2    2    0    4    5    1
%rej   6.6  3.3  3.1  0.0  6.5  9.3  1.7

total number of vectors = 422
overall correct   389 for  92.18%
overall error      15 for   3.55%
overall reject     18 for   4.27%
                                                                  Current
                                                                  Option·
                                                                  fisher

                                                                  Dataset:
                                                                  nasatest
                                                                  ****
```

Figure 1-23: Confusion Matrix for Temporary Within-Group Logic

1.3.3.3.3 Confusion Matrix for Overall Logic Evaluation

The confusion matrix produced by overall logic evaluation (logicevl) is identical in format to the matrix produced by the temporary within-group logic routines nmv, fisher, and closedcn. If the user chooses to list this matrix on the high-speed printer, a list of all incorrectly classified vectors may be produced in the format described previously.

1.3.3.3.3.1 Reassociated Names

Additional flexibility is made possible for the overall logic evaluation of an independent data set by the use of the reassociated names capability. Utilizing the routine logicevl, any data set may be tested against logic designed on any other data set of equal dimensionality. However, the totals and percentages correct listed below the matrix will be useful only if the names of the data classes on which logic was designed are the same as the names of the data classes being evaluated. This may be accomplished through the use of reasname, which allows the user to tag logic nodes with any reassociated names.

In a case where two or more logic nodes have been given the same reassociated name, the totals below the matrix are formed by adding the confusion matrix entries for these logic nodes.

If reassociated names have been added to the logic tree, logicevl asks the user whether the reassociated names are to be used in the confusion matrix printout. If the response is yes, the reassociated names will be used in place of the original design names; if more than one logic node has the same reassociated name, only one entry will appear in the confusion matrix for that name. In all cases where reassociated names have been added to a logic tree, they will be used to determine whether the vectors in the data set being evaluated have been assigned correctly.

One further embellishment has been added to the reassociated name capability. If sense switch 2 is set prior to overall logic evaluation, the test of correctness is simply made on the display symbols of the classes involved, rather than on the entire four-character names.

1.3.4 Boolean Partitions

MOOS has also implemented a user capability for Boolean defined partitions of the feature space. This capability can be used in structure analysis, group logic and pairwise logic.

This is implemented through utilization of the PL/1 compiler under MULTICS. As a result of the flexibility of MULTICS, the analyst can write any Boolean statement (one that can be evaluated as true or false), provided that it is a legal PL/1 statement and that it conforms to certain conventions for referencing feature vector components, and then use that statement as the basis for a transformation or a partition.

1.3.5 Measurement Transformations

In addition to a measurement reduction transformation (trnsform) performed in conjunction with measurement evaluation computations, a data set within MOOS may be transformed by any of the following three independent transformations; normxfrm, eigentrn, or measxfrm. Upon execution of any of these algorithms, every vector in the selected data set is transformed and a new tree is created from the transformed vectors. The new tree will have the same structure as the original tree.

1.3.5.1 The Normalization Transformation

The normalization transformation, normxfrm, determines the standard deviation along each coordinate measurement of the selected data set. Each vector component within the data set is then modified by dividing it by its corresponding standard deviation. The resulting normalized data set will have unit variance along each coordinate measurement.

In some cases, normalization may be necessary to ensure that the various numerical calculations performed by MOOS (e.g. matrix inversions) are sufficiently accurate.

1.3.5.2 The Eigenvector Transformation

The eigenvector transformation, eigentrn, computes the eigenvectors of the covariance matrix of the selected data set (see Section 1.3.2). The user is then given the option of mapping the selected L-dimensional data set onto an M-dimensional eigenvector subspace ($M \leq L$) by selecting the M eigenvectors corresponding to the M largest eigenvalues. The resulting M-dimensional subspace provides a least squares fit to the selected data set, since the sum of the squared residual distances from the subspace is minimized. The error in fitting the data is given by summing the remaining eigenvalues:

$$\text{Squared Fitting Error} = \sum_{j=M+1}^{L} \lambda_j$$

The transformation essentially involves an orthonormal rotation of the basis vectors of the data set until they are aligned with the eigenvectors.

This technique has proven useful both as a research tool and as an aid to structure analysis and logic design. Measurement reduction may also be performed through use of the eigenvector transformation.

1.3.5.3 The Measurement Compiler Transformation

By using the routine measxfrm, the MOOS user may define new features which are functions of the original L measurements. The capability of the MULTICS PL/1 compiler is utilized in that any statements allowed by PL/1 may be used for this transformation.

The measurement compiler option provides the MOOS user with a practically unlimited capability for defining both linear and nonlinear transformations. Once the new features have been defined, the system will execute the transformation, thereby generating a new data tree whose vectors have the new user-defined features as their components.

1.3.5.4 Measurement Reduction Transformations

The MOOS system provides three methods for selecting a projection of the "current data" onto a coordinate subspace in conjunction with the three methods for evaluating the discriminatory value of each measurement (Section 1.3.1). Each of these measurement evaluation algorithms (dscrmeas, probconf, features) produces rank order displays of the L measurements according to a user-specified criterion. The user may select specified measurements from the data set via the commands sel$ _ _ _ and un$ _ _ _. The measurements which are chosen for retention define the coordinate subspace and the desired linear transformation. The user then calls the measurement reduction transformation routine trnsform to implement the specified transformation, thereby creating a tree identical in structure, but containing vectors of fewer measurements than the original data tree.

REFERENCES

1. Kanal, Laveen N., "Interactive Pattern Analysis and Classification Systems: A Survey and Commentary", IEEE Proceedings, Vol 60, No. 10, pp. 1200-1215, October 1972.

2. Sammon, J.W. Jr., "Interactive Pattern Analysis and Classification", IEEE Transactions on Computers, Vol C-19, pp. 594-616, July 1970.

3. Simmons, E.J. Jr., "Interactive Pattern Recognition - A Designers Tool", AFIPS Conference Proceedings, Vol 42, pp. 479-483, June 1973.

4. Marill, T. and Green, D.M. "On the Effectiveness of Receptors in Recognition Systems," IEEE Transactions on Information Theory, Vol. IT-9, pp. 11-17, January 1963.

5. Kadota, T.T. and Shepp, L.A. "On the Best Finite Set of Linear Observables for Discriminating Two Gaussian Signals," IEEE Transactions on Information Theory, Vol. IT-13, pp. 278-284, April 1967.

6. Sammon, J.W. Jr., "An Optimal Discriminant Plane", IEEE Transactions on Computers, Vol. C-19, pp. 826-829, September 1970.

SECTION 2

MOOS USER'S MANUAL

2.1 GENERAL REMARKS

This manual contains descriptions of all operating MOOS user functions. It is designed to provide a potential user of the system with sufficient information to allow functional utilization of the system capabilities, but does not contain tutorial information as to the purpose underlying the development of each algorithm, nor does it contain details of the computations performed by any system program. Such information has been documented elsewhere.

The standard terminal from which MOOS commands are executed is the Tektronix 4002A storage tube display interfaced to the Honeywell 6180 MULTICS processor. The MULTICS control language has been utilized to the fullest possible extent in the development of MOOS, and therefore a working knowledge of the MULTICS environment is essential. For further information, consult the MULTICS Programming Manual. User function calls are input via the console keyboard and consist of simple program names (normally up to eight characters in length) followed by any required or optional parameters. Within system programs, dialogue concerning additional information required for program operation is handled by standard terminal input/output operations as specified within this manual.

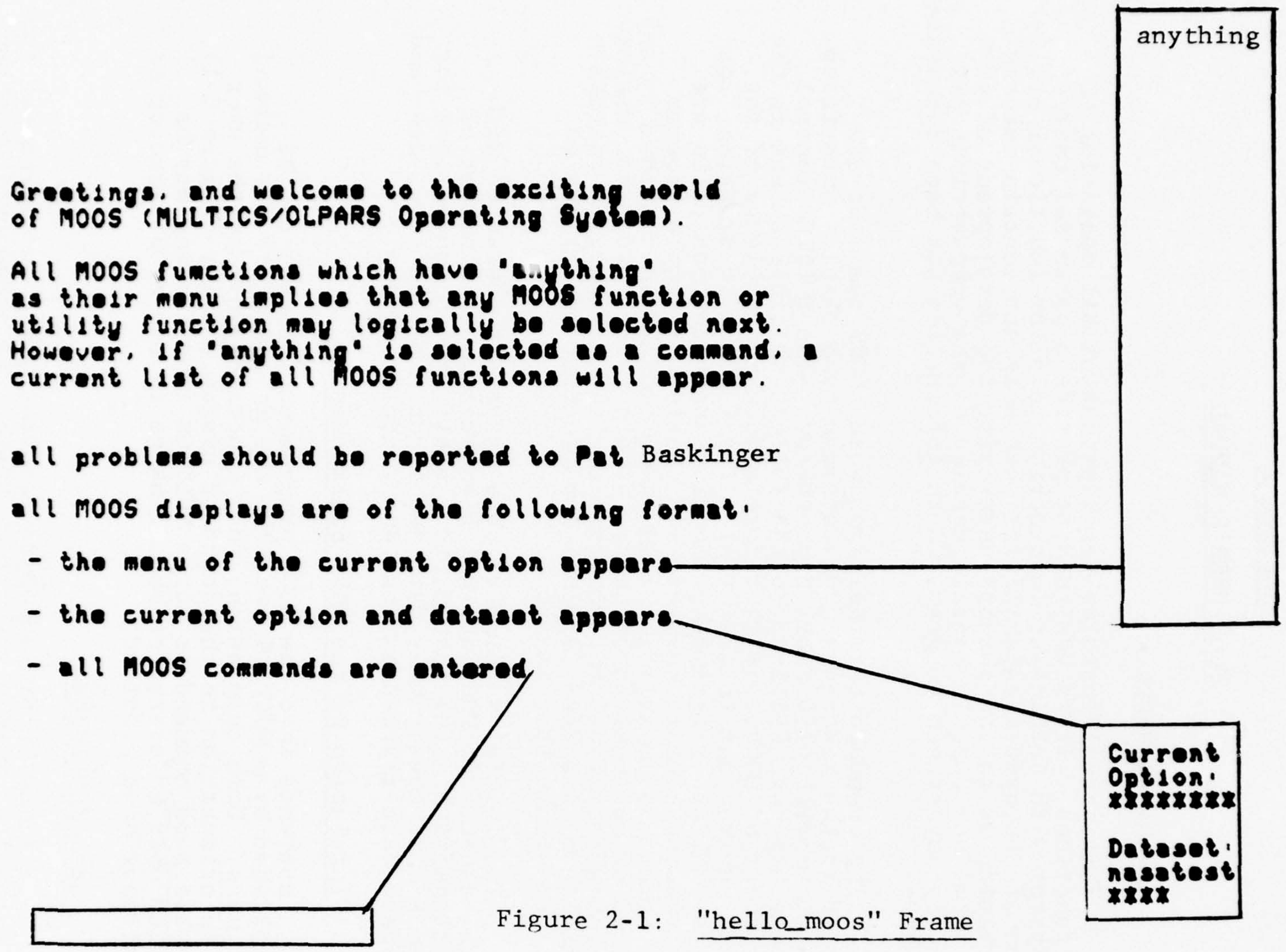
The initial section of this manual is concerned with several aspects of MOOS use which are common to a number of operations within the system, i.e., basic system conventions for entrance into the system, data set input and selection, one- and two-space data representation, and cursor movement.

Initiation of the MOOS Environment

Entrance into the MOOS program environment can be accomplished by a MULTICS user via the execution of the command hello_moos. Upon completion, this function provides the user with an orientation to the standard system display (Figure 2-1). The hello_moos command is not a MULTICS function and may be utilized only by users for whom linkage to the MOOS directories has already been provided.

Data Set Input

Data may be brought into active storage and formatted for MOOS usage in a variety of ways. Currently, procedures have been implemented which will accept data from cards (crdinput), magnetic tapes (tapinput), and three data file formats (fileinput,



```
Greetings, and welcome to the exciting world
of MOOS (MULTICS/OLPARS Operating System).

All MOOS functions which have 'anything'
as their menu implies that any MOOS function or
utility function may logically be selected next.
However, if 'anything' is selected as a command, a
current list of all MOOS functions will appear.


all problems should be reported to Pat Baskinger

all MOOS displays are of the following format:

 - the menu of the current option appears

 - the current option and dataset appears

 - all MOOS commands are entered
```

Figure 2-1: "hello_moos" Frame

restore, restorec). Data input programs must be called with a single parameter (five to eight characters in length) which represents the tree name of the input data set in the MOOS system.

The Standard Data Set Selection Parameters

Each of the major user programs under MOOS may be called by the user with up to two optional parameters, which represent the tree name and the name of the data class upon which the operation is to be performed. Rules for these parameters follow:

o All tree names are required to be five to eight characters in length. Data class names are four characters in length (the final character is used as a display symbol).

o If two parameters are input, the first will represent the tree name. For data input routines, the tree name must be unique (not currently maintained as a tree name within the system). For all other MOOS calls, the tree name input must currently exist within the system. The second parameter must be a legal data class name within the selected data tree.

o If one parameter is input, it may represent either a tree name or a data class name. If is is five to eight characters in length, it is taken to be a tree name, and the current data class is set by default to the senior node (symbolized by "****"). A *four-character* input is assumed to be a data class name within the current data tree.

o If no parameters are input, the current data tree and class parameters are not changed and operations of the called function are executed upon the same data set as the previous option.

Data Representation

MOOS provides the user with the capability of projecting a data set into a one- or two-space representation. Programs which produce data displays of these types have been given names of the form pppp$ffn, where

pppp	- is a four-letter code designating the type of projection plane ("eigv" - eigenvector; "crdv" - coordinate vector; "asdg" - assigned discriminant grouping; "ardg" - arbitrary discriminant grouping; "gndv" - generalized discriminant vectors).
ff	- is a code specifying the type of function to be performed ("sa" - structure analysis; "ld" - logic design).

n - is the number of vectors to be used in creating the data presentation ("1"; "2").

Thus, for example, the command eigv$sal allows the user to select one eigenvector and utilizes the current data set to produce a histogram under the structure analysis module.

Both one-and two-space data representations are available in two forms at the option of the user:

o One-Space Macro (Figure 2-2)

An unconfounded view of up to 16 data classes in a "stack histogram" format.

o One-Space Micro (Figure 2-3)

A view overlay of selected data class histograms represented in symbolic format (with a bar graph option).

o Two-Space Cluster (Figure 2-4)

Two-dimensional representation of the data which "forces" each data point into a location within a 36 x 60 grid. If one or more vectors from a single data class fall within the same grid location, the display symbol for that class is presented. If vectors from two or more data classes fall within a single grid location, an asterisk is displayed.

The two-space cluster display is generally faster than the two-space scatter display, especially for a large data set. However, since each character displayed may represent one or more vectors, in some cases this display could be misleading.

o Two-Space Scatter (Figure 2-5)

A two-dimensional representation within which each data point is located at its "natural" projection point.

Extensive facilities for manipulation and modification of the data projection displays are available; these are listed under Data Projection Modification Options in the program index below, and are described in detail in the appropriate user program listing.

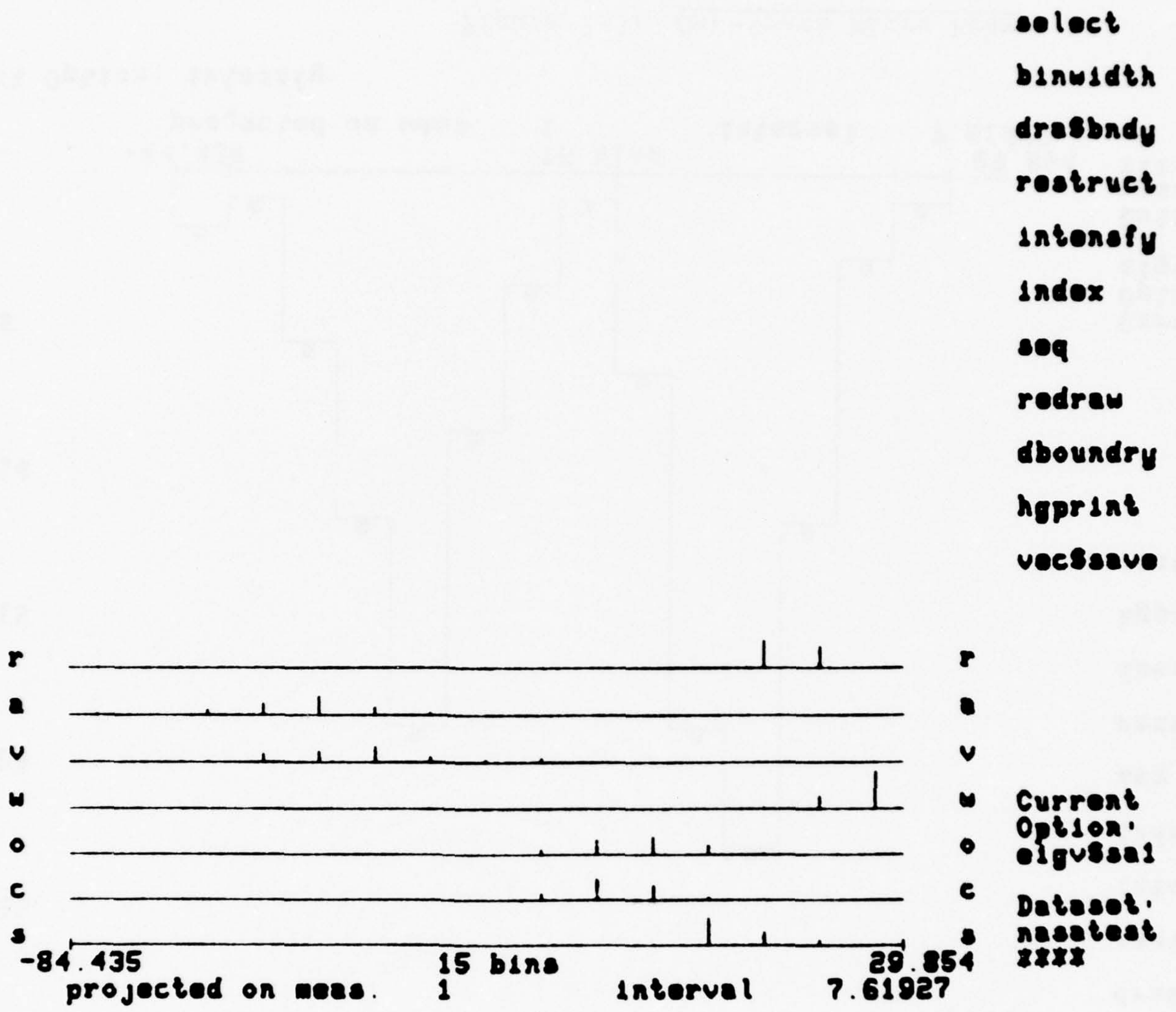
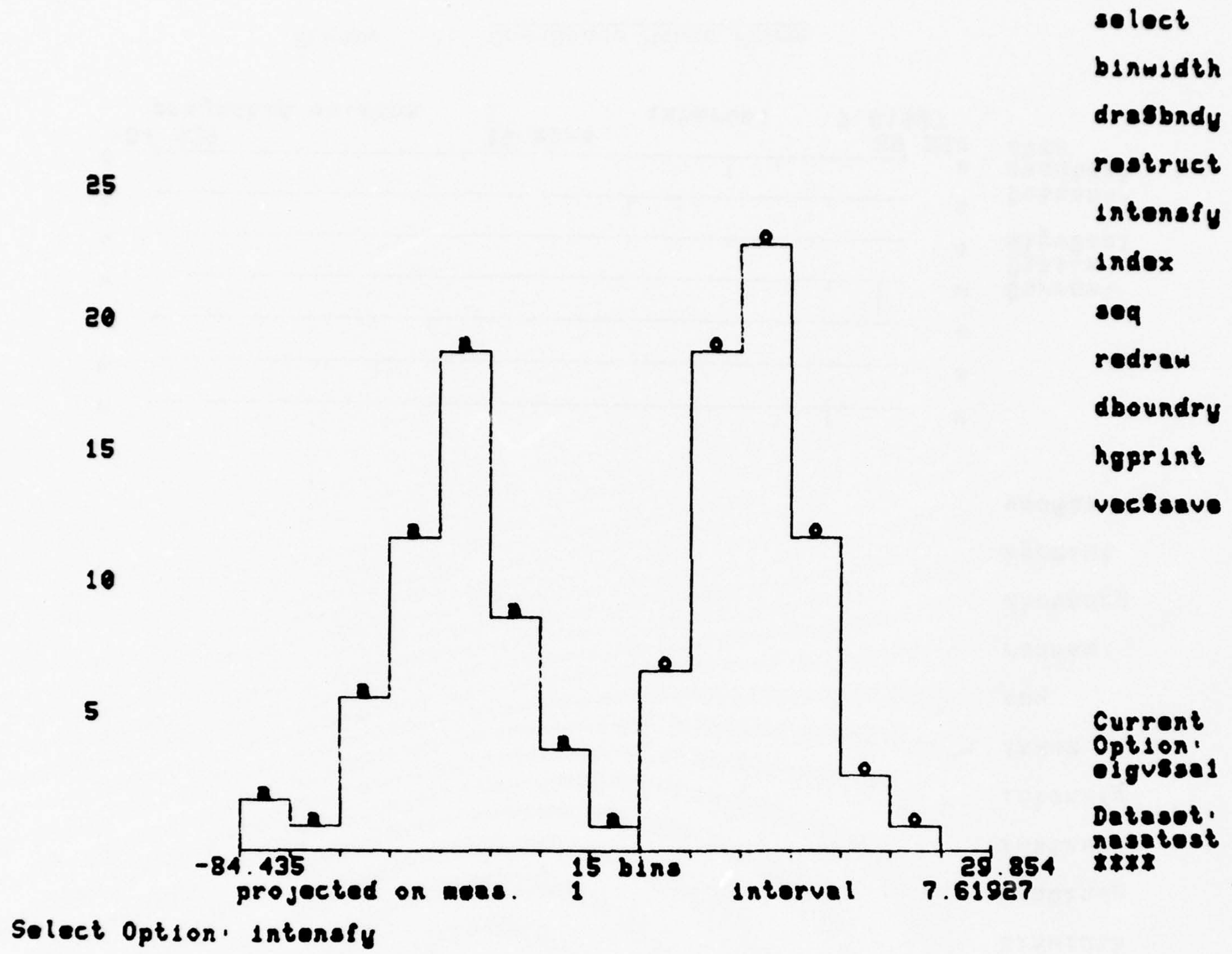
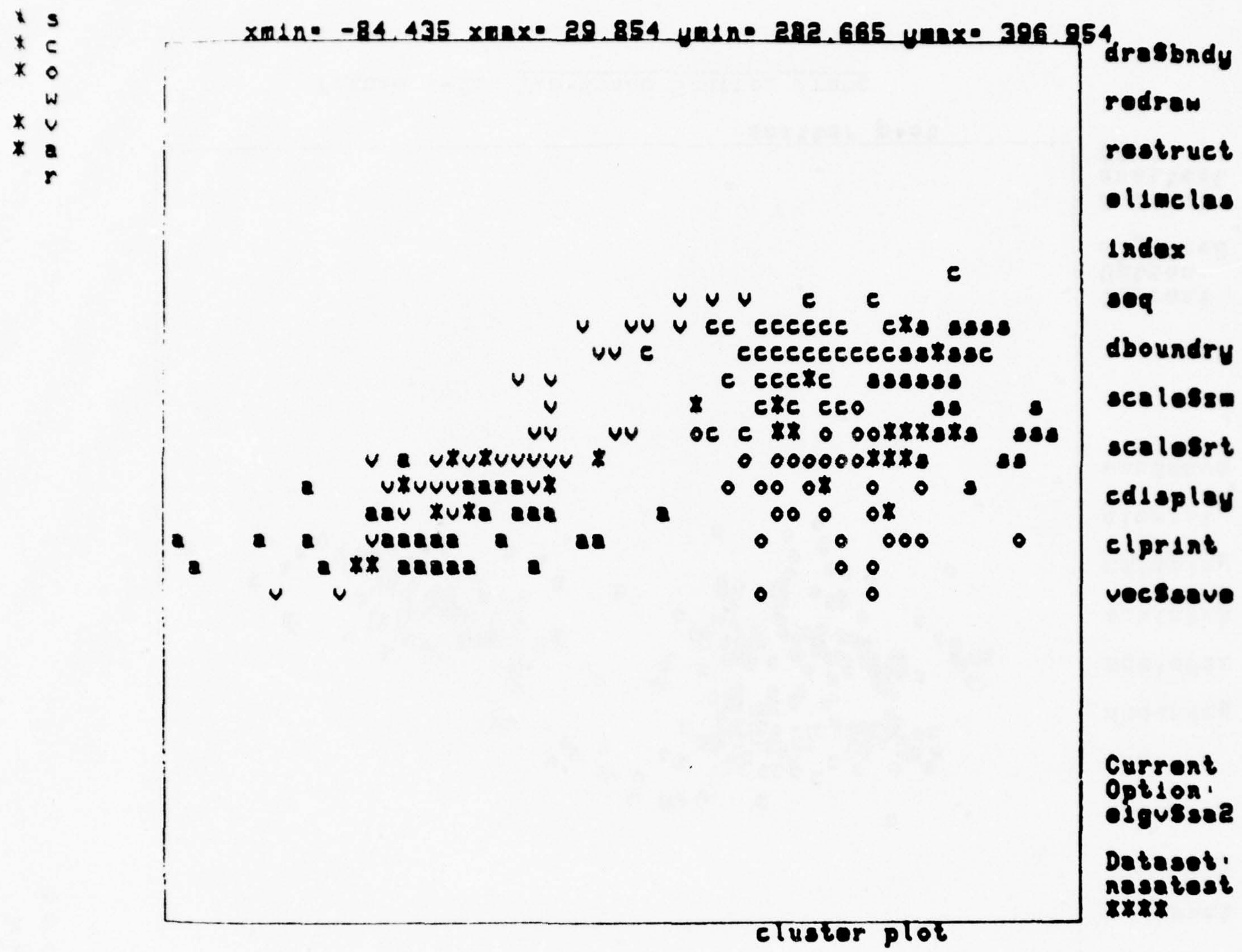


Figure 2-2: <u>One-Space Macro Frame</u>



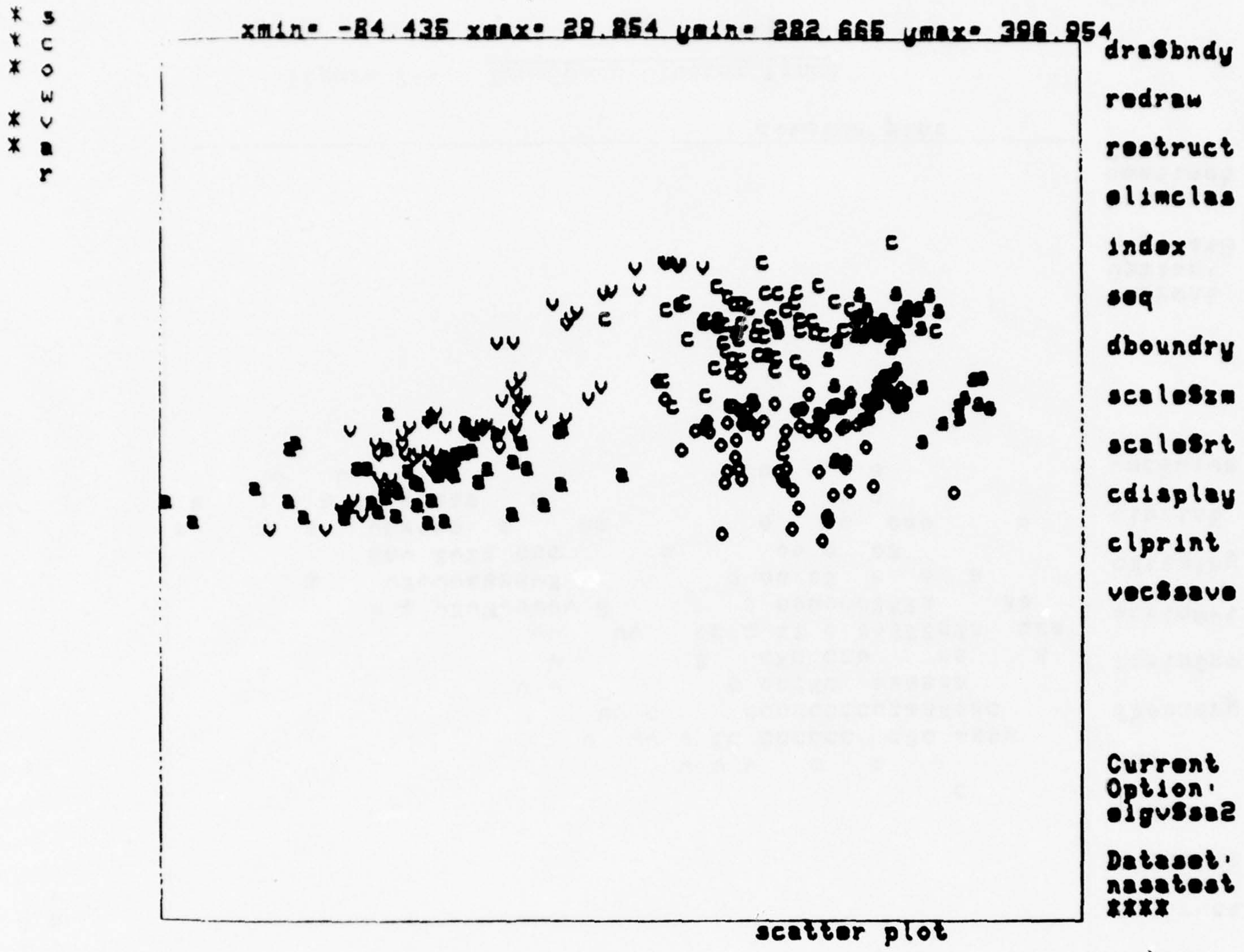
```
Current
Option:
eigv$sa1

Dataset:
nasatest
XXXX

Select Option: intensfy
```

Figure 2-3: <u>One-Space Micro Frame</u>



```
xmin= -84.435 xmax= 29.854 ymin= 282.665 ymax= 396.954
```

Figure 2-4: Two-Space Cluster Frame



```
xmin= -84.435 xmax= 29.854 ymin= 282.665 ymax= 396.954
```

Figure 2-5: <u>Two-Space Scatter Frame</u>

2.2 USER PROGRAM INDEX (by function)

The following index lists the current MOOS options by functional group:

SYSTEM UTILITY OPTIONS

User Call	Option Function	Page
anything	list current MOOS functions	2-26
crrandts	create a test data set from the current data tree	2-53
dataprnt	printout data characteristics and statistics	2-54
draw$_ _ _	display a logic tree	2-63
hello_moos	MOOS system entrance	2-82
rdisplay	reconstruct the latest one-space, two-space, or confusion matrix display	2-121
sense	set system sense switches	2-141
treedraw	display the current data tree	2-147
treelist	list the data trees in current active storage	2-148

DATA INPUT/OUTPUT OPTIONS

User Call	Option Function	Page
crdinput	input a data tree from punched cards	2-44
creatree	create a data tree from existing data classes	2-49
crrandts	create (extract) a test data tree from the current data tree	2-53
eigentrn	create a data tree via an eigenvector transformation	2-69
fileinput	input a data tree from MULTICS data file	2-76
measxfrm	create a data tree via Boolean (linguistic) transformation	2-103
mergmeas	create a data tree by combining the measurements of two existing trees	2-105
moosmode	convert from the excess measurement mode to normal MOOS operation	2-106
normxfrm	create a data tree via the normalization transformation	2-111
restore	input a data tree from exclusive user storage	2-125
restorec	input a data tree from common user storage	2-126
save	output a data tree to exclusive user storage	2-132
savec	output a data tree to common user storage	2-133
tapeoput	output a data tree to magnetic tape	2-145
tapinput	input a data tree from magnetic tape	2-146
trnsform	create a data tree via measurement reduction	2-149

DATA TREE MODIFICATION OPTIONS

User Call	Option Function	Page
append	add a data class to the current data tree from another existing data tree	2-27
chngname	modify tree name or data class name	2-37
comnod	combine data classes from the current data tree	2-43
creatree	create a data tree from existing data classes	2-49
crrandts	create (extract) a test data tree from the current data tree	2-53
deletnod	remove a data class from a data tree	2-58
deletree	remove a data tree from current data storage	2-59
dsubstrc	remove a subnode structure from a data tree	2-66
dvectors	remove data vectors from a data tree	2-68
lingpart	create a subnode structure via use of Boolean (linguistic) statements	2-92
restruct	create a subnode structure via partition of a data projection display	2-127
treedraw	display the current data tree	2-147
treelist	list the data trees in current data storage	2-148

DATA STORAGE OPTIONS (including housekeeping options for current data storage, common user storage, and exclusive user storage)

User Call	Option Function	Page
cleartree	remove a data tree from exclusive user storage	2-38
deletree	remove a data tree from current data storage	2-59
list_cst	list all data trees in common user storage	2-95
list_ust	list all data trees in exclusive user storage	2-96
log$_ _	input, output, or list MOOS logic in exclusive user storage	2-98
remtree	remove a data tree from common user storage	2-124
restore	input a data tree from exclusive user storage	2-125
restorec	input a data tree from common user storage	2-126
save	output a data tree to exclusive user storage	2-132
savec	output a data tree to common user storage	2-133
treelist	list the data tree in current data storage	2-148
vec$del	remove vector from exclusive user storage	2-152
vec$lall	list all vectors stored in exclusive user storage	2-152
vec$save	output projection vectors to exclusive user storage	2-153

DATA SET PROJECTION OPTIONS (all projection options with the exception of nlm are to be called via entry points indicating type of function ["sa" or "ld"] and represention space ["1" or "2"] ; e.g., "arbv$ld2": arbitrary vectors for logic design, two-space).

User Call	Option Function	Page
arbv$_ _ _	projection on arbitrary vectors	2-28
ardg$_ _ _	projection on arbitrary grouped discriminant plane	2-29
asdg$_ _ _	projection on assigned grouped discriminant plane	2-31
crdv$_ _ _	projection on coordinate vectors	2-47
eigv$_ _ _	projection on eigenvectors	2-70
fshp$_ _ _	projection on Fisher discriminant vectors	2-80
gndv$_ _ _	projection on generalized discriminant vectors	2-81
nlm	projection in nonlinear mapping space (structure analysis only)	2-107

DATA PROJECTION MODIFICATION OPTIONS

User Call	Option Function	Page
binwidth	modify bin size for one-space display	2-33
cdefault	change default values for data projections (bin size for one-space, data class size cutoff point for two-space cluster projection)	2-34
cdisplay	change two-space display format (cluster/scatter)	2-35
clprint	printout two-space cluster display	2-42
creatlog	create logic from one- and two-space projection partition(s) (logic design only)	2-48
dboundry	remove a partition from a data display	2-56
dra$bndy	draw a partition on a data projection	2-61
elimclas	change data class composition of the two-space data display	2-71
hgprint	printout one-space display	2-83
index	identify selected data points on data projections	2-87
intensfy	bargraph selected classes in one-space data display	2-88
redraw	redraw a previously implemented boundary	2-123
restruct	create a subnode structure via partition of a data projection display (structure analysis only)	2-127
scale$_ _	modify the scale factor for a data projection display	2-134
select	modify the one-space data presentation format (display format [macro/micro], data class composition, range, number of bins, and histogram type [count/probabilities])	2-138

DATA PROJECTION MODIFICATION OPTIONS (Continued)

User Call	Option Function	Page
seq	sequence for eigenvectors, coordinate vectors, or nonlinear map	2-142
vec$save	output projection vectors to exclusive user storage	2-153

STRUCTURE ANALYSIS OPTIONS (all data projections are appropriate for structure analysis; call desired data projection followed by "sa<u>n</u>" where <u>n</u> indicates one(1)- or two(2)-space projection)

User Call	Option Function	Page
lingpart	create a subnode structure via use of Boolean (linguistic) statements	2-92
restruct	create a subnode structure via partition of a data projection display	2-127

LOGIC DESIGN AND DEVELOPMENT OPTIONS (all data projections except nlm are appropriate for logic design; call desired data projection followed by "ldn" where n indicates one(1)- or two(2)-space projection)

User Call	Option Function	Page
chngaprb	modify a priori probabilities	2-36
closedcn	create closed decision boundary logic	2-39
closemod	modify a closed decision boundary logic node	2-40
creatlog	create logic from one-and two-space projection partition(s)	2-48
deletlog	remove node(s) from a logic tree	2-57
displacm	confusion matrix display	2-60
draw$_ _ _	display a logic tree	2-63
fisher	create Fisher pairwise logic	2-77
forteval	evaluate FORTRAN subroutine logic	2-78
fortlogc	create FORTRAN subroutine logic	2-79
hrdcpycm	confusion matrix printout	2-86
latclogc	create a "lattice type" logic tree structure	2-89
linglogc	create Boolean (linguistic) logic	2-91
lingrjct	create Boolean (linguistic) independent reject strategy	2-93
listlogc	printout logic tree	2-97
log$_ _	input, output, or list MOOS logic in exclusive user storage	2-98
logicevl	evaluate logic for completed logic design	2-102
nmv	create nearest mean vector logic	2-109
nmvmod	modify a nearest mean vector logic node	2-110
pairmod	modify a pairwise logic node	2-113
reasname	modify reassociated data class names in MOOS logic file	2-122
summrycm	confusion matrix summary display	2-144

MEASUREMENT EVALUATION OPTIONS

User Call	Option Function	Page
dscrmeas	discriminant measure measurement evaluation computation	2-65
features	"divergence measure" measurement evaluation computation	2-73
features_abs	enter an absentee request to execute the features algorithm off-line	2-75
histgram	display measurement evaluation in histogram format	2-84
hrdcpy	printout measurement evaluation listings	2-85
page	page rank order display	2-112
probconf	probability of confusion measurement evaluation computation	2-120
rnk$_ _	rank measurements for selected class	2-128
sel$_ _	select measurements for measurement reduction	2-136
trnsform	create a data tree via measurement reduction	2-149
un$_ _	select measurements for measurement reduction (union best class and/or class pair)	2-150

DATA TREE TRANSFORMATION OPTIONS

eigentrn	create a data tree via an eigenvector transformation	2-69
measxfrm	create a data tree via Boolean (linguistic) transformation	2-103
normxfrm	create a data tree via the normalization transformation	2-111
trnsform	create a data tree via measurement reduction	2-149

DATA PRINTOUT OPTIONS

User Call	Option Function	Page
clprint	printout two-space cluster display	2-42
dataprnt	printout data characteristics and statistics	2-54
hgprint	printout one-space display	2-83
hrdcpy	printout measurement evaluation listings	2-85
hrdcpycm	printout confusion matrix	2-86
listlogc	printout logic tree	2-97
vec$hall	printout saved projection vectors	2-152

PROGRAMMER AID OPTIONS

User Call	Option Function	Page
dump	printout standard system information	2-67
fastdump	printout selected system file information	2-72
setdata	set selected system file information	2-143

2.3 USER PROGRAM INDEX (Alphabetical)

Name	Option Function	Page
anything	list current MOOS functions	2-26
append	add a data class to the current data tree from another existing data tree	2-27
arbv$_ _ _	projection on arbitrary vectors	2-28
ardg$_ _ _	projection on arbitrary grouped discriminant plane	2-29
asdg$_ _ _	projection on assigned grouped discriminant plane	2-31
binwidth	modify bin size for one-space display	2-33
cdefault	change default values for one-and two-space data projections	2-34
cdisplay	change two-space display format	2-35
chngaprb	modify a priori probabilities	2-36
chngname	modify tree name or data class name	2-37
cleartree	remove a data tree from exclusive user storage	2-38
closedcn	create closed decision boundary logic	2-39
closemod	modify a closed decision boundary logic node	2-40
clprint	printout two-space cluster display	2-42
commod	combine data classes for the current data tree	2-43
crdinput	input a data tree from punched cards	2-44
crdv$_ _ _	projection on coordinate vectors	2-47
creatlog	create logic from one-and two-space projection partition(s)	2-48
creatree	create a data tree from existing data classes	2-49
crrandts	create (extract) a test data set from the current data tree	2-53

(Continued)

Name	Option Function	Page
dataprnt	printout data characteristics and statistics	2-54
dboundry	remove a partition from a data display	2-56
deletlog	remove node(s) from a logic tree	2-57
deletnod	delete node from a current data tree	2-58
deletree	remove data tree from current data storage	2-59
displacm	confusion matrix display	2-60
dra$bndy	draw a partition on a data projection	2-61
draw$_ _ _	display a logic tree	2-63
dscrmeas	discriminant measure evaluation computation	2-65
dsubstrc	remove a subnode structure from a data tree	2-66
dump	printout standard system information	2-67
dvectors	remove data vectors from a data tree	2-68
eigentrn	create a data tree via an eigenvector transformation	2-69
eigv$_ _ _	projection on eigenvectors	2-70
elimclas	change data class composition of the two-space data display	2-71
fastdump	printout selected system file information	2-72
features	"divergence measure" measurement evaluation computation	2-73
features_abs	enter an absentee request to execute the features algorithm off-line	2-75
fileinput	input a data tree from a MULTICS data file	2-76
fisher	create Fisher pairwise logic	2-77
forteval	evaluate FORTRAN subroutine logic	2-78
fortlogc	create FORTRAN subroutine logic	2-79
fshp$_ _ _	projection on Fisher discriminant vectors	2-80
gndv$_ _ _	projection on generalized discriminant vectors	2-81
hello_moos	MOOS system entrance	2-82

(Continued)

Name	Option Function	Page
hgprint	printout one-space display	2-83
histgram	display measurement evaluation in histogram format	2-84
hrdcpy	printout measurement evaluation listings	2-85
hrdcpycm	printout confusion matrix	2-86
index	identify selected data points in data projections	2-87
intensfy	bargraph selected classes in one-space data display	2-88
latclogc	create a "lattice type" logic tree structure	2-89
linglogc	create Boolean (linguistic) logic	2-91
lingpart	create a subnode structure via use of Boolean (linguistic) statements	2-92
lingrjct	create Boolean (linguistic) independent reject strategy	2-93
list_cst	list all data trees in common user storage	2-95
list_ust	list all data trees in exclusive user storage	2-96
listlogc	printout logic tree	2-97
log$_ _ _	input, output, or list MOOS logic in exclusive user storage	2-98
logicevl	evaluate logic for completed logic design	2-102
measxfrm	create a data tree via Boolean (linguistic) transformation	2-103
mergmeas	create a data tree by combining the measurements of two existing data trees	2-105
moosmode	convert from the excess measurement mode to normal MOOS operation	2-106
nlm	projection in nonlinear mapping space	2-107
nmv	create nearest mean vector logic	2-109
nmvmod	modify nearest mean vector logic node	2-110
normxfrm	create a data tree via the normalization transformation	2-111

AD-A034 393   PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y   F/G 9/2

MULTICS OLPARS OPERATING SYSTEM. VOLUME I.(U)

SEP 76   D B CONNELL, K N KLINGBAIL   F30602-75-C-0226

UNCLASSIFIED   PAR-74-25-A   RADC-TR-76-271-VOL-1   NL

2 OF 3

AD
A034393

(Continued)

Name	Option Function	Page
page	page rank order display	2-112
pairmod	modify pairwise logic node	2-113
probconf	probability of confusion measurement evaluation computation	2-120
rdisplay	redraw latest one- or two-space display	2-121
reasname	modify reassociated data class names	2-122
redraw	redraw previously implemented boundary	2-123
remtree	remove a data tree from common user storage	2-124
restore	input a data tree from exclusive user storage	2-125
restorec	input a data tree from common user storage	2-126
restruct	create a subnode structure via partition of a data projection display	2-127
rnk$_ _ _	rank measurements for selected class	2-128
save	output a data tree to exclusive user storage	2-132
savec	output a data tree to common user storage	2-133
scale$_ _	modify the scale factor for a data projection display	2-134
sel$_ _ _	select measurements for measurement reduction	2-136
select	modify the one-space data presentation format	2-138
sense	set system sense switches	2-141
seq	sequence eigenvectors, coordinate vectors, or nonlinear mapping	2-142
setdata	set selected system file information	2-143

(Continued)

Name	Option Function	Page
summrycm	confusion matrix summary display	2-144
tapeoput	output a data tree to magnetic tape	2-145
tapinput	input a data tree from magnetic tape	2-146
treedraw	display the current data tree	2-147
treelist	list the data trees in current data storage	2-148
trnsform	create a data tree via measurement reduction	2-149
un$_ _ _ _	select measurements for measurement reduction	2-150
vec$_ _ _ _	input, retrieve, remove, or list projection vectors in exclusive user storage	2-152

Function Call: anything

Additional User Interaction: None

Function Description: anything gives a list of all currently implemented MOOS functions and all user-callable routines.

Function Call: append tree1 node1 tree2 node2 newnode

Input Parameters:

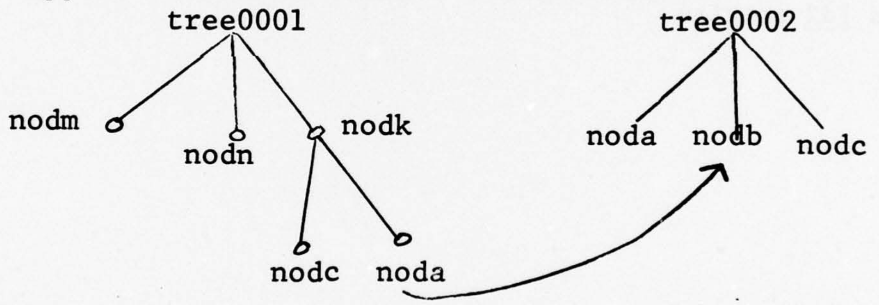
tree1	source data tree
node1	source data node
tree2	destination data tree
node2	destination data node
newnode	name of node created

Additional User Interaction: None

Function Description: append has two similar functions: 1) it combines two nodes; or 2) it transfers a node from a source tree to a destination tree and calls it newnode. When a user wishes to combine two nodes, only the first four parameters are required; node2 must be a lowest node. If a user wants to copy a node, then all five parameters are needed and newnode must be a unique node name under tree2. The node2 parameter will serve as the superior node for newnode. Tree1 and tree2 can be the same tree.

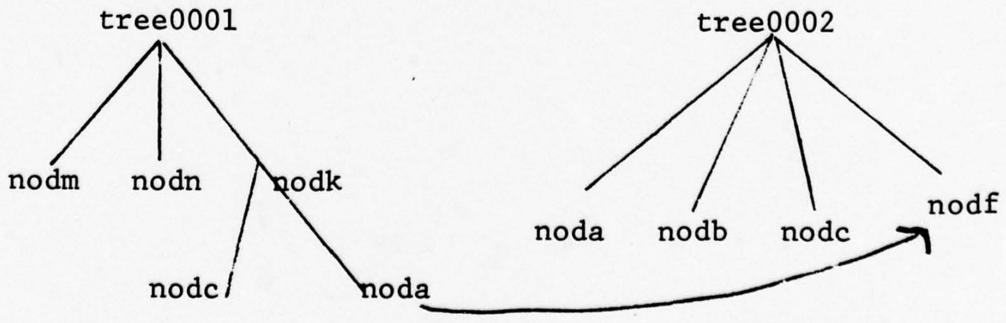
Example 1: To combine noda of tree0001 with nodb of tree0002, the input sequence is

"append tree0001 noda tree0002 nodb"



Example 2: To attach noda of tree0001 under the senior node of tree0002 and call it nodf, the calling sequence is

"append tree0001 noda tree0002 **** nodf"



Function Call: arbv$sa1 (treename) (classname)
arbv$sa2 (treename) (classname)
arbv$ld1 (treename) (classname)
arbv$ld2 (treename) (classname)

Additional User Interaction: Do you want to retrieve a saved projection vector, or type in a new projection vector?

```
Type in s or i
s
Enter the name of a projection vector or type "stop"
vector name
```

Function Description: arbv projects the selected data set on any vector or vectors chosen by the user. The projection vector(s) may be manually input, or retrieved from the user created saved vectors file (see function vec$save).

Entries sa1 and sa2 of arbv are for use in structure analysis. The ld1 and ld2 entries are used in designing logic.

The sa1 and ld1 entries of arbv generate one-space or histogram plots of the data. Two-space plots are created by the sa2 and ld2 entries.

Function Call:

```
ardg$sa1  (treename)  (classname)
ardg$sa2  (treename)  (classname)
ardg$ld1  (treename)  (classname)
ardg$ld2  (treename)  (classname)
```

Additional User Interaction: Choose classes from the following list:

```
class(1)  class(2)  ...  class(n)
```

Type in the number of classes in group 1 followed by each class name.

```
One entry to a line
number
class(i)
```

Type in the number of classes in group 2 followed by each class name.

```
One entry to a line
number
class(j)
class(k)
```

Enter one or two options from the following list: (on one line - no delimiters)

```
0     default options (covariance matrix and all measurements)
1     scatter matrix
2     eliminate some measurements
```

```
number
```

Type in c to display results

```
c
```

Function Description: ardg projects the selected data set on the Fisher direction (or the optimal discriminant plane) associated with two user-assigned groupings of data classes. These groupings need not comprise the entire data set; however, the entire data set is projected on the resulting Fisher vector(s).

The calculation of the Fisher direction may be based on the sum of the within-group scatter matrices rather than the sum of the within-group covariance matrices by selecting option 1. If option 2 is chosen, the Fisher direction may be computed using a subset of the feature space.

Entries sa1 and sa2 of ardg are for use in structure analysis. The ld1 and ld2 entries are used in logic design.

The sa1 and ld1 entries of ardg generate one-space or histogram plots of the data. Two-space plots are created by the sa2 and ld2 entries.

Function Call:

```
asdg$sa1   (treename)  (classname)
asdg$sa2   (treename)  (classname)
asdg$ld1   (treename)  (classname)
asdg$ld2   (treename)  (classname)
```

Additional User Interaction:

```
classes in group 1 are:
class(1) ... class(i)

classes in group 2 are:
class(i+1) ... class(n)

do you wish to modify these groupings?
no

Enter one or two options from the following
list:  (on one line - no delimiters)

0   default options (covariance
    matrix and all measurements)
1   scatter matrix
2   eliminate some measurements

number

type in c to display results
c
```

Function Description: asdg projects the selected data set on the Fisher direction (or the optimal discriminant plane) associated with two algebraically-assigned groupings of data classes. The assignment of groups is carried out in the following manner: First, the two data classes whose mean vectors have the largest Euclidean separation are found. The remaining data classes are then associated with the class of this pair to which they are closest. If the user is not satisfied with the assignment of groups, he may modify the groupings manually or use ardg, which expects the user to input groups of his own. The final groupings need not comprise the entire data set; however, the entire data set is projected on the resulting Fisher vector(s).

The calculation of the Fisher direction may be based on the sum of the within-group scatter matrices rather than the sum of the within-group covariance matrices by selecting option 1. If option 2 is chosen, the Fisher direction may be computed using a subset of the feature space.

Entries sa1 and sa2 of asdg are for use in structure analysis. The ld1 and ld2 entries are used in logic design.

The _sal_ and _ldl_ entries of _asdg_ generate one-space or histogram plots of the data. Two-space plots are created by the _sa2_ and _ld2_ entries.

Function Call: binwidth

Additional User Interaction: There are three questions whose answers determine how the display will be modified.

Function Description: The user may modify the current one-space display. Using binwidth, the changes that can be made are:

1) changing the xmin or starting point of the display,
2) changing the number of bins, and
3) changing the interval size

The three questions are:

1) "enter new xmin or 'same'"
2) "enter new number of bins or have it vary as a function of the interval size by entering 'vary'"
3) "enter new interval size"

For #1, the user replies either with the new starting point (bin number) or with "same." For #2, the user replies either with the new number of bins or with "vary." For #3, the user enters the new interval size. The number of bins will be affected unless the current number of bins is entered in response to #2; if that occurs, the xmax or end point of the display is adjusted. If *all these questions* are answered with new values, the xmax of the display is again altered.

Function Call: cdefault

Additional User Interaction: Select changes from the following list:

1) cluster/scatter cutoff value (default value: 500)
2) one-space bin factor (default value: 5)
3) return to initial default values
4) no changes

number of an option

Function Description: Option 1: If the number of vectors in a data set to be projected on a two-space plot is greater than the cutoff value, a cluster plot is displayed; otherwise a scatter plot is shown. If option 1 is selected, the program requests that a new value be input. Example: If the cutoff value were set to zero, all two-space plots would initially be displayed in the cluster mode.

Option 2: The initial number of bins for a histogram plot is determined by dividing the total number of vectors in the data set by the number of classes times the one-space bin factor. If option 2 is selected, the program requests that a new value be entered. Example: If the number of classes is 1, the bin factor is set to 2, and the total number of vectors is 100, these vectors will be placed in 100/(1*2) = 50 bins.

Option 3: If option 3 is selected, the cluster/scatter cutoff value is set to 500 and the one-space bin factor is set to 5.

Function Call: cdisplay

Additional User Interaction: None

Program Description: If the current display is a two-space scatter plot, cdisplay changes the display to a two-space cluster plot and vice versa.

Function Call: chngaprb (treename) (classname)

Additional User Interaction: Upon initiation, the program lists the a priori probabilities for treename and classname, and the user must enter one of the following options:

0 - a priori probabilities will not be changed from those listed
1 - make all a priori probabilities equal
2 - enter proportion for each class

Function Description: chngaprb allows a user to change the a priori probabilities.

Option 1: the a priori probability for each class is set equal to 1/ncls where ncls is the number of classes.

Option 2: the user is asked to enter a weighting value for each class. For class (i), the a priori probability equals $\left[1 \div \sum_{j=1}^{ncls} wt\ (j)\right] wt\ (i)$, where $\sum_{j=1}^{ncls} wt(j)$ is the sum of all the entered proportions, and wt (i) is the proportion entered for class (i).

After the user selects an option, the amended list is presented, and the user must input another option.

Option 0 will exit him from the routine; the last values presented are the a priori probabilities.

If more than 16 classes are to be presented, the user will be asked to "enter any number to proceed." The number entered will simply clear the screen, and will not be used to calculate any probabilities.

Function Call: chngname (treename) (classname)

Additional User Interaction: Input number of names to be changed

number

Input on separate lines each name to be changed followed by a new name

old name (1)
new name (1)
:
:
:
old name (n)
new name (n)

Function Description: chngname permits the user to change the name of any MOOS data tree or the names of any nodes within a data tree. Any new names must be unique within the specified tree. The last characters of any new node names must also be unique, since they are used as display characters.

Function Call: cleartree treename/"all"

Input Parameters:

treename	specify a particular data set
"all"	remove all existing trees from the user's "saved_trees" directory

Additional User Interaction: None

Function Description: This routine deletes from the user's "saved_trees" directory trees and their associated nodes that have been copied via the utility function save.

Function Call: closedcn (treename) (classname)

Additional User Interaction:

Is the type of closed decision boundary logic to be the same for all classes?

yes

```
Enter 1 for hyperrectangle
      2 for hypersphere
      3 for hyperellipsoid
```

2

** A partial evaluation of the logic is generated and displayed.

Function Description:

closedcn creates closed decision boundary logic at a user-specified logic node. Closed decision boundary logic attempts to enclose each of the classes in the selected data set with a closed n-dimensional hyperregion (n = number of dimensions). There are three types of hyperregions available for this purpose: hyperrectangles, hyperspheres, and hyper-ellipsoids. The user has the option of specifying one of these hyperregion types for all the classes in the selected data set, or he may choose a different hyperregion for each class. In the case of a hyperrectangular surface the user may also select one of the following basic vector types: the coordinate vectors of the data set, the eigenvectors of the covariance matrix of the entire data set, or the eigenvectors of the covariance matrices of the individual data classes.

Any sample vectors rejected by the evaluation of the closed decision boundary logic created by closedcn are rejected due to "overlap." This means that the vector lies within more than one closed decision boundary*. The user may create a new data tree from vectors rejected due to overlap for further logic design.

* Boundaries created by closedcn are always large enough to include all the vectors in the selected data set.

Function Call: closemod (treename) (classname)

Additional User Interaction:

```
Choose a class from the following list:
ab...f
b

Current logic type for class b is hyperrectangular

Select a logic type
1.  hyperrectangular
2.  hyperspherical
3.  hyperellipsoid
1

Select an option:
a.  Default thresholds (based on the range of the data)
b.  Thresholds based on % of the range of the data
c.  Display and interactive modification of the thresholds
d.  New basis vectors

d

Do you wish to modify logic for another class?

no

**  A partial evaluation of the logic is generated and
displayed.
```

Function Description:

closemod may perform a number of modifications to a previously created closed decision boundary logic. It is possible using this routine to change the type of boundary associated with a class, or to modify the parameters which specify the current closed decision boundary logic for a class. It should be noted that the user has more control over the setting of boundary parameters than with the original logic creation program closedcn.

As with the closedcn program, the user may choose to create a new data tree from sample vectors which fall into more than one closed decision boundary. Vectors which do not lie within any closed decision boundary are rejected and may not be placed in the new data tree.

The options allowed for each logic type are listed below:

- For hyperrectangular logic the user may choose any one of the following:

  a. Default thresholds (based on the range of the data)
  b. Thresholds based on % of the range of the data
  c. Display and interactive modification of the thresholds
  d. New basis vectors

- For hyperspherical logic the user may choose 1 or 2 of the following:

  a. Default center vector (mean of the class)
  b. Center vector at median of the class
  c. User-input arbitrary center vector
  d. Default radius (based on the range of the data)
  e. Radius based on % of range of the data
  f. User input radius

- For hyperellipsoid logic the user may choose 1, 2, or 3 of the following:

  a. Default center vector (mean of the class)
  b. Center vector at median of the class
  c. User-input arbitrary center vector
  d. Default axis lengths (based on range of the data)
  e. Axis lengths based on % of default axis lengths
  f. Display and interactive modification of axis lengths
  g. Mahalanobis (axis lengths = eigenvalues)
  h. Default "C" value (based on the range of the data)
  i. "C" value based on % of default "C" value

(The "C" value refers to a quantity analogous to the radius of a hypersphere)

Note: A large amount of error checking is performed on the above-mentioned option lists to ensure that certain oversights on the part of the user are avoided (e.g., the user may not change the radius of a hyperspherical logic by more than one method at a time, etc.) Also, when a user selects a different logic type for a class than what is currently implemented, any options not selected which are needed to specify the logic are set to their default value.

Function Call: clprint

Additional User Interaction: None

Function Description: A copy of the latest two-space cluster plot is produced on the high-speed printer.

If the current display is a scatter plot and a hardcopy is desired, the user must first change to a cluster plot via cdisplay.

Function Call: comnod (treename) (classname)

Additional User Interaction: You may combine a maximum of n nodes from the following list:

node (1) node (2) ... node (n+1)

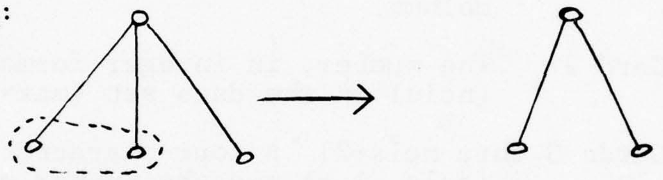
Enter list of nodes to be combined:

abc...m

input new 4 character node name

new name

Function Description:



comnod is used for combining data nodes under a common intermediate node (or the senior node) in a MOOS data tree. comnod will not allow all the data nodes under a common node to combine into a single node; dsubstrc should be used for this purpose.

The new four-character node name of the combined node, as well as its display character, must be unique in the specified tree.

Function Call: crdinput treename

Additional User Interaction: None

Function Description: The crdinput function transforms a data set input on cards into a MOOS data tree. The input parameter treename must be unique in the current system. All data parameters must be included on the data cards as described below. The card data must be stored in the user's file "testdata" and may be input to the system as follows:

o The data deck consists of ncls+2 cards which describe the data, followed by the data cards.

  Card 1: The number, in integer format, of dimensions (ndim) of the data set (max=100) in any column.

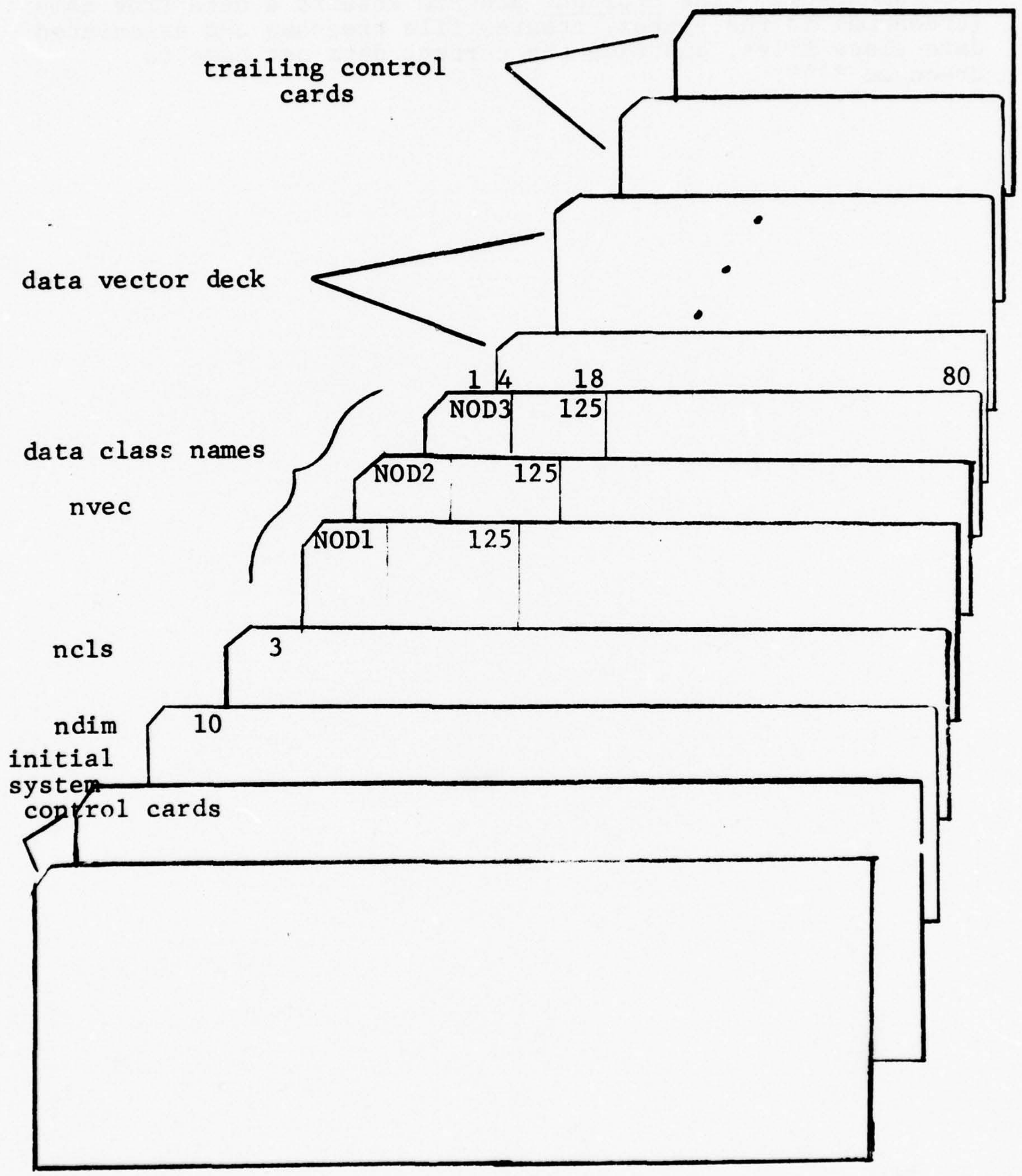
  Card 2: The number, in integer format, of classes (ncls) in the data set (max=72) in any column.

  Cards 3 thru ncls+2: A four-character data class name (cols. 1-4) and the number of vectors within this data class (nvec) in integer format, right-adjusted to column 18.

  The data vector deck: the data measurement values in floating point format (no assumed decimal point) with at least one space between each value. Index values for each vector are assigned sequentially by crdinput. No extraneous information may appear on the cards.

o The standard system control cards must be added to the front and the end of the data deck.

The following example illustrates a data deck consisting of three classes (nod1, nod2, nod3). Each class contains 125 ten-dimensional vectors.


trailing control cards
data vector deck
1
4
18
80
NOD3
125
data class names
NOD2
125
nvec
NOD1
125
ncls
3
ndim
10
initial system control cards

Program Result: The crdinput program inserts a data tree name (treename) in the system, creates file treename and associated data class files, and sets the current data set name to treename ****.

Function Call: crdv$sa1 (treename) (classname)
crdv$sa2 (treename) (classname)
crdv$1d1 (treename) (classname)
crdv$1d2 (treename) (classname)

Additional User Interaction: input the 2 coordinates onto which the data is to be projected.

number1 number2

Function Description: crdv projects the selected data set onto coordinate axes chosen by the user.

Entries sa1 and sa2 of crdv are for use in structure analysis. The 1d1 and 1d2 entries are used in designing logic.

The sa1 and 1d1 entries of crdv generate one-space or histogram plots of the data. Two-space plots are created by the sa2 and 1d2 entries.

Function Call: creatlog

Additional User Interaction: The user must enter the display symbols of the data classes within each partition of the current display (one or more boundaries must have been previously drawn using dra$bndy). Each list of display symbols must be on one line with no delimiters. If four asterisks are entered, it is assumed that the given partition is to be a reject region.

The results of evaluating the data set on the newly created logic are presented in a confusion matrix format. The user is then given the option of listing the matrix on the high-speed printer. A listing of all incorrectly classified vectors may also be produced.

Function Description: creatlog creates and evaluates the logic for a boundary drawn in a one- or two-space plane. creatlog should be called after the use of any "ld1" or "ld2" projection plane program.

Function Call: createtree treename

Additional User Interaction: The user selects the mode of creating a new **tree (tre**ename) by specifying the number of nodes:

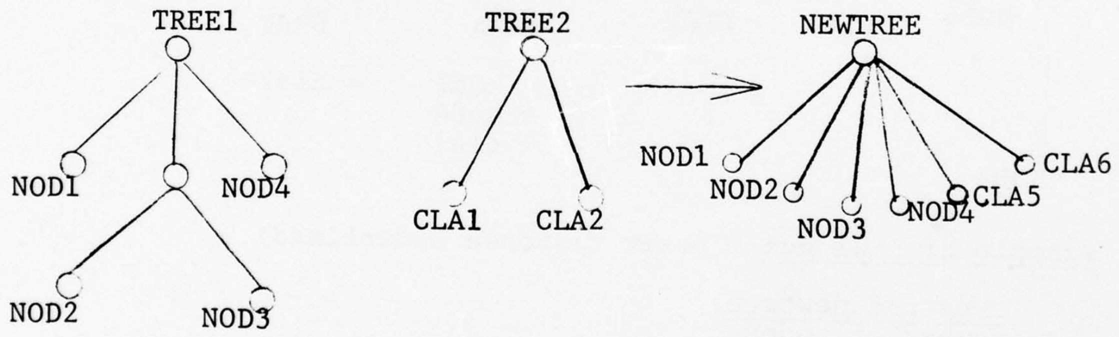
combine trees - enter a number < 0
merge trees - enter 0
user-specified composition - enter number of nodes to be made.

The user then specifies up to ten trees to be used in creating treename, expresses whether vector ID's are to be sequenced, and indicates whether a listing of sequence changes is desired.

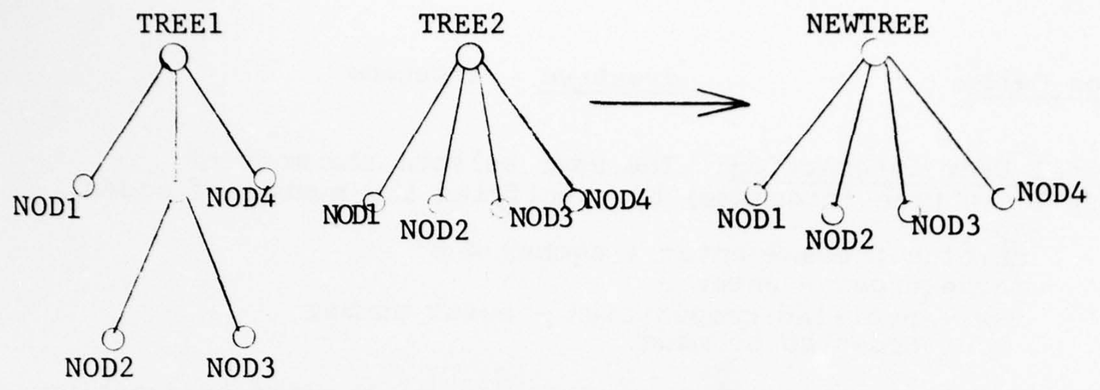
Further dialogue is needed in the case of composition specification, where the user designates which nodes (if any) from each of the specified data trees will be used to create the treename nodes.

Function Description:

Combine trees: enter number of nodes < 0.
The routine will create treename with copies of the lowest nodes of the up to ten trees. In the process, if any display symbols are not unique, new display symbols will be appended. The total number of nodes must be less than or equal to 72.

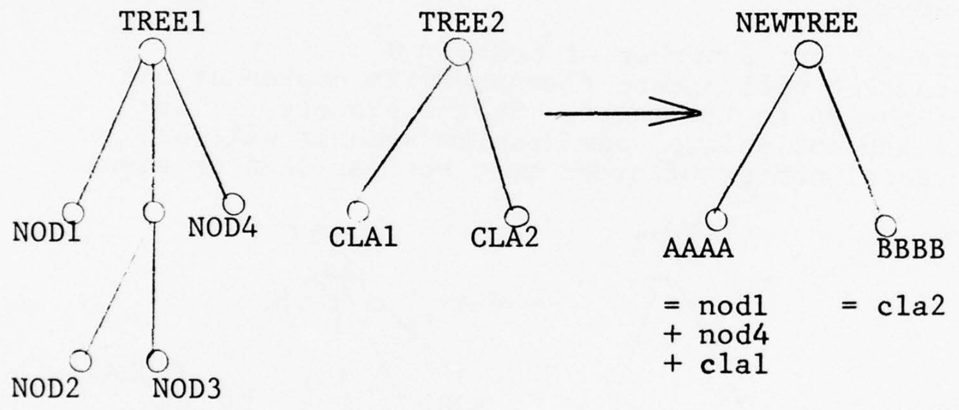


Merge trees: enter number of nodes = 0
The routine will create treename by merging all of the vectors from similarly-named nodes to the treename node of the same name. All of the up to ten trees must have the same number of lowest nodes, and these lowest nodes must have the same names.



User-specified composition: enter number of nodes to be made.

The user will specify each treename node name and the number and names of nodes to be merged from the up to ten trees. If the new node name does not have a unique display symbol, the user is asked to input another name for the new node. If a particular node is not found, the user has the option to exit or enter a new node.



Example of dialogue: (user response underlined)

```
creatree newtree
number of nodes
2
number of trees used in making new tree
2
names of these trees, one per line
tree1
tree2
do you wish to sequence vector ID numbers as the vectors
   are input?
```

```
no
(end of dialogue if number of nodes ≤0)
enter a four-character name for new nodel of tree newtree
AAAA
enter list of display symbols of nodes from tree treel
to be used in forming node AAAA--enter on one line with
no delimiters (enter ** if no nodes are to be used)
14
number of nodes from tree treel = 2, using nodes:
nodl nod4
enter list of display symbols of nodes from tree tree2 to
be used in forming node AAAA--enter on one line with no
delimiters (enter ** if no nodes are to be used)
1
number of nodes from tree tree2 = 1, using node clal
enter a four-character name for new node2 of tree newtree
BBBA
display character is not unique
reenter new four-character name for node2
BBBB
enter list of display symbols of nodes from tree treel
to be used in forming node BBBB - enter on one line with
no delimiters (enter ** if no nodes are to be used)
**
number of nodes from tree treel = 0
enter list of display symbols of nodes from tree tree2
to be used in forming node BBBB - enter on one line
with no delimiters (enter ** if no nodes are to be used)
2X
number of nodes from tree2 = 2, using nodes cla2
no node exists with display symbol X
do you wish to reenter a new list of display symbols of
nodes from tree2 to be used for node BBBB of tree
newtree?
yes
enter new list from tree tree2 for node BBBB
2
number of nodes from tree tree2 = 1, using node cla2
BEGIN PROCESSING
```

If the user specifies one node to be created for newtree, the new four-character name is automatically set to "****".

The option to sequence vector ID numbers should be used in a situation where two or more trees have vectors whose ID numbers are the same. A listing of ID number changes may be produced at this time if the user wishes.

Possible errors:

creatree will exit if the dimensions of the up to ten trees are not equal, if any of the up to ten trees are not known to the system, or if the user does not wish to continue after entering an incorrect list of nodes. In these cases, the routine will delete the newtree and all the new nodes created thus far, and will return with the current data set "notatree," "nono."

Function Call: crrandts new-treename

Additional User Interaction: enter tree name where vectors are to be extracted from.

eight-character tree name
enter percent of data to be extracted
number [for example, if 50% of the data are to be extracted, enter 50 rather than .5]

Function Description: crrandts is a means of creating a random test set. A user-selected percentage of vectors from the given data tree are removed and placed in a new tree.

Function Call: dataprnt (treename) (classname)

Additional User Interaction: Type in options from the following list:

```
1.  all vectors
2.  single vector
3.  ranges & overlap
4.  means & standard deviations
5.  difference between mean vectors
6.  covariance matrices
7.  correlation matrices
8.  tree structure
e   exponential format of output
c   change nodes to be processed
134ec8

6 options selected
BEGIN PROCESSING

1 request signalled, 0 already queued
```

Function Description: dataprnt outputs certain basic information about a data set to the high-speed printer. A functional description of each option follows:

o 1. all vectors All the vectors in the selected data set are printed. The format of this printout simplifies comparison of the different values of a specific measurement or feature.

o 2. single vector If option 2 is selected, the program asks for a specific vector ID number and prints the corresponding vector. The program then asks if more single vectors are desired. This option is useful for printing a limited number of individual vectors.

o 3. ranges & overlap This option prints a table for each data class in the selected data set, containing the minimum, maximum, and range of each measurement. A table containing minimum, maximum, and range values for the entire data set is also printed. Following these tables is a list of pairs of classes which do not overlap along each specific measurement.

o 4. means & standard deviations The mean vectors and standard deviations for every data class and for every intermediate node in the selected tree are printed.

o <u>5. difference between mean vectors</u> This option prints the Euclidean distance between each pair of mean vectors in the selected data set, and also the absolute values of the differences between each measurement of these mean vectors.

o <u>6. covariance matrices</u> The covariance matrix for every data class and for every intermediate node in the selected tree structure is printed.

o <u>7. correlation matrices</u> The correlation matrix for every data class and for every intermediate node in the selected tree structure is printed.

o <u>8. tree structure</u> The tree structure of the selected data set is printed in outline form, including the number of vectors at each node.

o <u>e exponential format of output</u> Exponential format may be used any time the user prefers to have values printed in scientific notation. This may have a distinct advantage in cases where more than five-digit accuracy is desired or where the values of the measurements are greater than 10^4 or less than 10^{-4}. The regular format is usually easier to read, however.

o <u>c change nodes to be processed</u> If this option is selected, questions are asked which allow the user to specify any subset of the selected data set to be processed by <u>dataprnt</u>. For example, if the user had originally chosen a data set with ten lowest data class nodes and seven intermediate nodes, he could change and get printouts which only involved the ten lowest nodes.

<u>Note</u>: If any unusual errors occur while <u>dataprnt</u> is executing, the user will find himself in his "login" directory.

Function Call: dboundry

Additional User Interaction: None

Function Description: Any thresholds or boundaries which have been created in current one- or two-space plots are deleted.

<u>Function Call</u>: deletlog (treename) (classname)

<u>Additional User Interaction</u>: Upon initiation, the program displays the current logic tree.

Select a logic node to be deleted (0 indicates quit)
<u>logic node number</u>

The logic at node (logic node number), and all nodes below it, has been deleted.

<u>Function Description</u>: <u>deletlog</u> deletes a designated logic node and all logic nodes below it in a logic tree. If logic node 1 is selected, all logic is deleted for the selected tree.

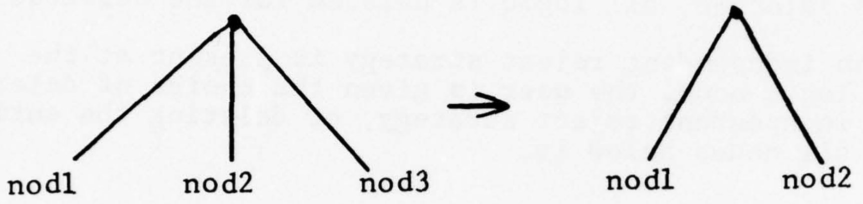
If an independent reject strategy is present at the selected logic node, the user is given the choice of deleting only the independent reject strategy, or deleting the entire node and all nodes below it.

Function Call: deletnod (treename) (classname)

Additional User Interaction: None

Function Description: deletnod causes the entered class name (assuming that it is a lowest node) to be deleted from the given tree. If the entered class name is not a lowest node, an error message will be printed.

Example: If the current data tree consists of three data classes: nod1, nod2, nod3, and "deletnod nod3" is entered, all data associated with nod3 will be removed from temporary storage.



Note: Upon completion the current data set is changed to treename****.

Function Call: **deletree** treename/"all"

Input Parameters:

treename	specify a particular data set
"all"	remove all existing trees in the process directory.

Additional User Interaction: None

Function Description: This routine deletes any or all trees and their associated nodes from "sysdata" file. The user is cautioned against the use of the parameter "all" unless the data have been saved via the utility functions save or savec.

<u>Function Call</u>: <u>displacm</u>

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>displacm</u> outputs all confusion matrix information to the screen, including the numbers and percentages of vectors correct, in error and rejected.

Function Call: dra$bndy

Additional User Interaction: Boundaries in a two-space display are drawn as follows:

After the routine is initiated, a crosshair will appear on the screen. The crosshair may now be moved to any point on the screen. When the crosshair is in the desired location, one of three characters is entered ["c", "e", or "q"]. The "c" (continue) means that more points for this boundary are to be read. The "e" (end) means that this is the end of the first boundary, but another boundary is to be drawn. The "q" (quit) means no more points for any boundaries are to be read (i.e. this is the end of all boundary drawing). The crosshair is then turned back on and this whole process is repeated. After two points are read, a line segment will appear between these two points. For subsequent points, a line is drawn from the end (i.e. second point) of the previous line segment to the new point. After the last line segment for each boundary is drawn, the crosshair (hereafter referred to as x-hair) is turned on once again, and the user moves it to the convex side of the boundary just drawn. Any character can now be entered and is read as a point on the convex side of the boundary.

As an example, let us suppose that two boundaries with boundary 1 having three line segments, and boundary 2 having two line segments, are to be drawn. The following steps should be followed:

1. type in dra$bndy.
   - x-hair displayed

2. move x-hair to 1st point, enter a "c ".
   - x-hair redisplayed

3. move x-hair to 2nd point, enter a "c ".
   - a line segment is drawn from point 1 to point 2, x-hair redisplayed

4. move to 3rd point, enter a "c ".
   - a line is drawn from point 2 to 3, x-hair redisplayed

5. move to 4th point, enter an "e ".
   - a line is drawn from point 3 to 4, x-hair redisplayed

6. move to convex side of first boundary, enter any character.
   - x-hair redisplayed

7. move to 1st point of 2nd boundary, enter a "c ".
   - x-hair redisplayed

8. move to 2nd point of 2nd boundary, enter a "c ".
   - a line is drawn from point 1 to 2 of 2nd boundary, x-hair redisplayed

9. move to 3rd point of 2nd boundary, enter a "q ".
   - a line is drawn from point 2 to 3 of 2nd boundary, x-hair redisplayed

10. move to convex side of 2nd boundary, enter any character.
   -the routine is terminated

Let us suppose that only one boundary was to be drawn, with three line segments. In this case, the first six steps of the above procedure would be followed, except that in step 5, a "q" would be entered rather than an "e ".

Boundaries on a one-space display are drawn as follows:

The crosshair is positioned to the desired point on the screen and either a "q" or an "e" is entered. The "q" as in a two-space boundary, means quit, i.e. no further boundaries are intended. The "e" is end and signifies that a second threshold is to be drawn. No convex points are selected.

Function Description: dra$bndy allows for drawing up to two boundaries (maximum of five line segments per boundary) on the display after a two-space plot is put on the screen, or up to two thresholds after a histogram has been plotted.

Function Call: draw$log (treename) (classname)

Additional User Interaction: None

Function Description: draw$log produces a pictorial display of a user-created logic at any stage in the development of the logic. Logic nodes are displayed with interconnecting lines to illustrate their relationship to each other. To the right of all incomplete or lowest logic nodes is a list of the classes present at that node. If there is an independent Boolean reject strategy associated with a logic node, an arrow appears above that node.

If a given logic tree is too large to display on the screen, a message is printed and as much as possible of the tree is displayed. In this case, extra lines are drawn to indicate what part of the structure is missing from the display. Any portion of a logic tree may be viewed by using draw$prt.

Function Call: draw$prt (treename) (classname)

Additional User Interaction:

input a node number
number

Function Description: draw$prt produces a pictorial display of a portion of a user-created logic at any stage in the development of the logic. The selected logic node and all logic nodes below it are presented in the same format as draw$log. Logic nodes are displayed with interconnecting lines to illustrate their relationship to each other. To the right of all incomplete or lowest logic nodes is a list of the classes present at that node. If there is an independent Boolean reject strategy associated with a logic node, an arrow appears above that node.

If the selected portion of the logic tree is too large to display on the screen, a message is printed and as much as possible of the tree is displayed. In this case, extra lines are drawn to indicate what part of the structure is missing from the display.

Function Call: dscrmeas (treename) (classname)

Additional User Interaction: None

Function Description: dscrmeas computes the discriminant measurement evaluation statistics and outputs to the screen an overall ranking of the measurements for the current data set. The user may then select any of several ranking options to decide which measurements are best for separating particular classes (see documentation on rnk). The user may then select specified measurements from his data set via functions sel$meas and sel$thres and may perform a measurement reduction on the data set via function trnsform.

Possible errors:

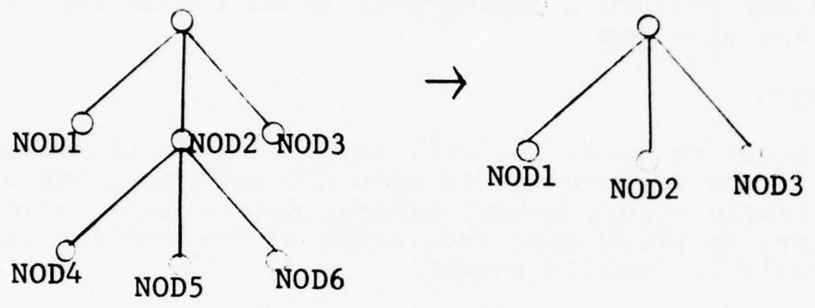
An exception condition will occur if (due to roundoff) the variance of any measurement is zero for more than one data class. This rarely occurs unless several data classes contain only one vector, in which case the result of any statistical calculation would be invalid anyway.

Function Call: dsubstrc (treename) (nodename)

Additional User Interaction: if the current data class is **** (the senior node), the program will ask the user if he wishes to make this the only node in the tree.

Function Description: dsubstrc permits a user to delete the substructure of an intermediate node. All the vectors under nodename are merged into nodename, which then becomes a lowest node.

Example: dsubstrc nod2



Function Call: dvectors (treename) (classname)

Additional User Interaction:

The user is asked if he wants to delete:

(1) all vectors
(2) a range of vectors
(3) one vector
number

If (2), an initial and last vector ID are requested to be entered.

If (3), a vector ID is requested to be entered

If (2) or (3) is entered and the last vector is being deleted from the class, the user is informed of this, and he is asked if he still wants to delete it.

Function Description: dvectors allows the user to delete all vectors, a range of vectors, or one vector from a given data class.

Function Call: dump

Additional User Interaction: None

Function Description: dump formats and outputs the following information to the line printer:

1) the first five words of "sysdata" (unpacked)
2) all forest entries which are not "notatree" in "sysdata", with all values unpacked
3) all school entries which are found by internal subroutine lnodes in "sysdata", with all values unpacked
4) TREENAME file information
   a) first word unpacked
   b) the list of lowest nodes
   c) the last index of the file for fastdump
5) for each DATACLASS file
   a) the number of vectors
   b) the last index of the file for fastdump

Function Call: eigentrn (treename)

Additional User Interaction:

Do you want a high-speed printout of eigenvalues?
no

**At this point, a list of eigenvalues for the selected data set is presented in descending order.

input the threshold eigenvalue
value

input a new treename
treename

Function Description: eigentrn generates a new tree of equal or lower dimensionality by transforming the selected data set, using the eigenvectors which correspond to the selected eigenvalues (the selected eigenvalues consist of the threshold eigenvalue and all eigenvalues above it in the list).

The transformation is done as follows:

$$nm(i) = \sum_{j=1}^{ndim} OM_j \times EV(i)_j$$

where $nm(i)$ = the ith measurement or feature in the new tree

OM_j = a component of a vector in the selected tree

$EV(i)_j$ = a component of the ith eigenvector

ndim = dimensionality of selected data set

Function Call:

```
eigv$sa1 (treename) (classname)
eigv$sa2 (treename) (classname)
eigv$sa1 (treename) (classname)
eigv$sa2 (treename) (classname)
```

Additional User Interaction:

Do you want a high-speed printout of eigenvalues?
no

**At this point, the eigenvalues for the selected data set are listed in descending order.

select an eigenvalue
number

Function Description: eigv projects the selected data set on an eigenvector or eigenvectors of that data set. The user chooses which eigenvector(s) he wants by choosing the corresponding eigenvalue(s).

After the one- or two-space plot appears on the screen, a different subset of the eigenvectors may be selected from the list of eigenvalues, which can be made to reappear by means of the option seq.

Entries sa1 and sa2 of eigv are for use in structure analysis. The 1d1 and 1d2 entries are used in designing logic.

The sa1 and 1d1 entries of eigv generate one-space or histogram plots of the data. Two-space plots are created by the sa2 and 1d2 entries.

Function Call: elimclas on (class1) (class2)... (class N)
elimclas off (class1) (class2)... (class N)

Additional User Interaction: None

Function Description: elimclas is the utility function which manipulates the classes currently displayed. It does not affect the true structure in any manner. The first of the parameters, which are separated by blanks, is either the character string "on" or "off". The other parameters are one-character class symbols. "On" indicates that the following parameters are to be displayed while "off" indicates that the following parameters are not to be displayed.

For example, assume a tree exists with 26 nodes, classes A, B ... Z; initially all the classes are displayed.

The following command would display classes A, F and S:

```
elimclas on A F S
```

The following command would display all classes except A, F AND S:

```
elimclas off A F S
```

The following command would display all classes, since the parameter list of classes to be turned off is null:

```
elimclas off
```

Function Call: fastdump filename

Additional User Interaction: The program requests the location of the file to be dumped, and asks what portion of the file is to be listed.

Function description: The program fastdump lists the value of each word in a given MULTICS file in six formats: integer, floating point, bit string, octal, character string (with blanks replacing unprintable characters), and exponential.

This dump may be sent either to the high-speed printer or to the user's terminal.

Function Call: features (treename) (classname)

Additional User Interaction:

```
Do you wish to do measurement selection interactively?
no
Do you wish to select any measurements to start with?
no

Enter type of measurement selection at each iteration.

1.  Pure forward sequential
    (one feature selected at a time - the highest in
    overall rank).
2.  un$bbcp approach.
3.  un$bbc approach.
    (one or more features selected at each iteration for
    options 2 and 3)
1
Enter maximum number of features to be selected.
10
```

Function Description:

features utilizes the divergence measurement evaluation criteria to aid the user in determining which measurements of the selected data set are most likely to be useful. features may be executed in an interactive mode or a non-interactive mode. Due to the lengthy nature of the calculations involved, the non-interactive mode may be preferable to the interactive mode for larger data sets. A MULTICS absentee job may be entered to perform a non-interactive execution of features by using function features_abs.

Interactive Mode:

The features program makes a number of passes over the set of measurements in the selected data set. After each pass, a rank order display is produced and one or more measurements may be selected by the user. All normal rank-order display options are allowed, including trnsform, which means that the user may create a number of data trees based on different subsets of the feature space for further analysis. At each pass, the user may choose to stop execution. If "stop" is chosen, a list of all selected measurements is presented.

Non-interactive Mode:

In the non-interactive mode, the features program performs measurement selection automatically and produces a detailed listing of the results. No results are sent to the screen.

As with the interactive mode, a number of passes are made over the set of measurements in the current data set. At each pass, one or more measurements are selected and this information is printed. The divergence values for each class pair and each measurement tested are also listed at each pass.

The measurement selection techniques available in the non-interactive mode are listed in the example (see write-ups on un$bbcp and un$bbc for a detailed description of their function). (Refer to Section 1.)

Function Call: features_abs

Additional User Interaction:

> Enter the treename of the data set whose features are to be evaluated:
> treename
>
> ** All further interaction mimics the user interaction of features.

Function Description:

features_abs creates and enters a MULTICS absentee job which executes the features routine in the non-interactive mode (see write-up on features).

The user interaction of features_abs is designed to mimic the interaction of features. The resulting output is identical to that which would be produced by an on-line execution of features.

In addition to the output generated by features, the ".absout" segment produced by the MULTICS absentee job is printed.

Function Call: fileinput treename

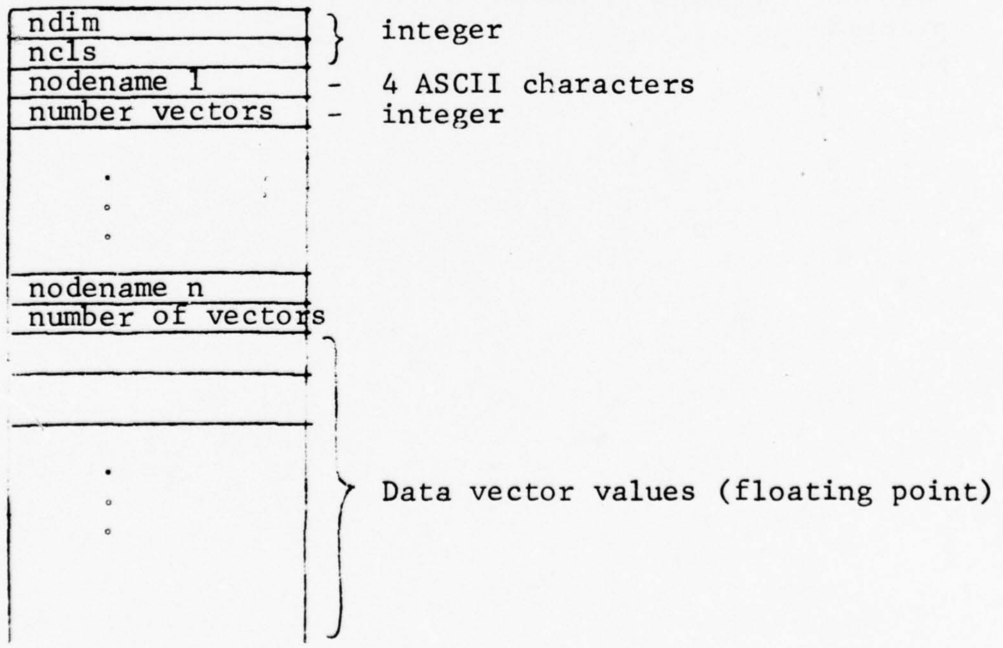
Additional User Interaction: The user will be requested to reenter the tree name if the name he has entered is greater than eight or less than five characters long, or if the tree name is already known to the system.

Function Description: This routine inputs a tree (treename) into the system, transforming the data contained in the file "filedata", which is located in his process directory.

"filedata" must be organized as described below:

Word 1: ndim - the number of dimensions
Word 2: ncls - the number of data classes
Word 3 thru 2(ncls)+2 - four-character data class name and number of data vectors within the data class; two words for each class.
Word 2(ncls)+3 thru end of the file - data vector values

It is up to the user to create the process directory file "filedata" from cards, tape, or another file.



Function Call: fisher (treename) (classname)

Additional User Interaction:

Enter one or two options from the following list: (on one line - no delimiters)

0	default options (covariance matrix and all measurements)
1	scatter matrix
2	eliminate some measurements

number(s)

Enter number of thresholds to be implemented
number (must be 1, 2, 3, or 4)

Enter minimum vote count (max = n)
[n = number of classes - 1]

number

[A confusion matrix display, as described in Section 1, is then presented.]

Function Description: fisher constructs and evaluates the Fisher pairwise discriminant logic as described in Section 1. It calculates the pairwise discriminant vector as well as the orthogonal discriminant vector for each pair of classes The latter is saved for possible future use. The five possible thresholds for each pair of classes are also calculated at this time. The Fisher discriminant may be calculated using either the sum of the scatter matrices or the sum of the covariance matrices; also, any measurements may be eliminated from the calculation. After the logic has been evaluated, the user may choose to change the number of thresholds or the minimum vote count and reevaluate the logic. More extensive modifications may be made by using pairmod.

<u>Function Call</u>: <u>forteval</u> (treename) (classname)

<u>Additional User Interaction</u>:

Input the name of the evaluation subroutine
<u>name</u>

** confusion matrix display is presented

<u>Function Description</u>:

<u>forteval</u> tests a selected data set against a FORTRAN subroutine generated by <u>fortlogc</u>. The resultant confusion matrix lists the number of vectors from each data class that were assigned to each logic node.

User-generated subroutines may also be evaluated using <u>forteval</u>. The subroutine must reside in the user's login directory, and have a six-character name. The parameter list of the subroutine must conform to the parameter list of a subroutine generated by <u>fortlogc</u>, although the language need not be FORTRAN. The first parameter must be the data vector to be evaluated, stored in an L-dimensional floating point array. The second parameter must be the assigned logic node number (integer).

Function Call: fortlogc (treename) (classname)

Additional User Interaction:

Enter one or more options (on one line - no delimiters)

a. Generate FORTRAN source program for punching
b. Generate and print FORTRAN listing
c. Generate FORTRAN object program for evaluation.

a

Enter a six-character subroutine name
name02

Source program name (for punching): name02.fortran

Function Description:

fortlogc generates a FORTRAN subroutine which can classify data vectors according to the logic strategy of a specific MOOS logic tree. The resultant FORTRAN source code may be punched on cards for use at other facilities (the generated source code is in "standard" FORTRAN and card format). The user may also request that the source code be compiled and listed, and/or an evaluation of the design data set produced. The subroutine created by fortlogc has two parameters: 1) an L-dimensional data vector (L-dimensional real array), 2) the assigned logic node number (integer).

A compiled FORTRAN subroutine may be used to evaluate any MOOS data set with the same dimensionality as the design data set through use of the MOOS function forteval.

Note: Some minor discrepancies between the results of overall evaluation (logicevl) and FORTRAN subroutine logic may occur. This is due to the difference between the internal representation of numbers and the decimal representation found in a FORTRAN subroutine generated by fortlogc. The evaluation of closed decision boundary logic in particular may be very sensitive on a design data set when the size of a boundary is based on the precise range of the data.

Function Call fshp$sa2 (treename) (classname)

fshp$1d2 (treename) (classname)

Additional User Interaction:

```
Fisher pairs may be chosen from the following list:
class(1) ...class(n)

enter first pair - one node name to a line
class(i)
class(j)

enter second pair
class(k)
class(l)

Enter 1 or 2 options from the following list:
(on one line - no delimiters)

0  default options (covariance matrix and all
   measurements
1  scatter matrix
2  eliminate some measurements
number(s)
```

Function Description: *fshp projects the selected* data set on two Fisher directions, which correspond to two pairs of data classes within the selected data set.

The calculation of the Fisher directions may be based on the sum of the within-class scatter matrices by selecting option 1. If option 2 is chosen, the Fisher directions may be computed using a subset of the feature space.

Entry sa2 of fshp generates a two-space plot for use in structure analysis.

Entry 1d2 generates a two-space plot for use in logic design.

Function Call:

```
gndv$sa1 (treename)   (classname)
gndv$sa2 (treename)   (classname)
gndv$ld1 (treename)   (classname)
gndv$ld2 (treename)   (classname)
```

Additional User Interaction:

** A list of eigenvalues is presented in descending order. Each eigenvalue corresponds to a generalized discriminant vector.

Select an eigenvalue

number

** One- or two-space display is presented.

Function Description:

gndv projects the selected data set on one or two of the generalized discriminant vectors associated with that data set. If the number of classes in the data set is denoted by n, then n-1 generalized discriminant vectors are calculated. The user may choose any of these vectors as projection vectors.

Entries sa1 and sa2 of gndv are for use in structure analysis. The ld1 and ld2 entries are for logic design.

The sa1 and ld1 entries of gndv generate one-space or histogram plots of the data. Two-space plots are created by the sa2 and ld2 entries.

If there are two classes in the selected data set, only one discriminant vector is calculated; therefore, only the sa1 and ld1 options may be chosen. gndv may not be executed on data sets with only one class.

<u>Function Call</u>: <u>hello_moos</u>

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>hello_moos</u> is the first option that should be selected upon entering MULTICS. Some introductory remarks and a description of MOOS will appear on the screen. After this routine is done, any option may then be selected.

<u>Function Call</u>: <u>hgprint</u>

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>hgprint</u> copies the latest one-space display to the printer. The last display must have been a "micro" view with less than 120 bins. If a copy of a display with more than 120 bins is desired, the user must call <u>select</u> and change the number of bins to less than 120.

<u>Function Call</u>:	<u>histgram</u> (classlist)

<u>Input parameters</u>:

classlist	This is an optional list of class symbols of classes to be displayed.

<u>Additional User Interaction</u>: The user must input the number of the measurement on which the data is to be projected.

<u>Program Description</u>: After executing the MOOS function <u>probconf</u>, the user can display the results on the console using <u>histgram</u>. If no class list is supplied, then all classes in the current data set are displayed. The user is asked to input the measurement number of the data he wishes to see. The results of projecting this data onto the designated feature are then displayed in the usual one-space format.

Function Call: hrdcpy

Additional User Interaction: None

Function Description: hrdcpy produces a copy on the high-speed printer of any desired rank order display for dscrmeas, probconf, or features.

Function Call: hrdcpycm

Additional User Interaction: The user is asked if he desires a listing of incorrectly classified vectors.

Function Description: hrdcpycm outputs all confusion matrix information to the line printer, including the numbers and percentages of vectors correct, in error and rejected. In addition, information about incorrectly assigned vectors may be output.

Function Call: index ("count"/"id")

Additional User Interaction: index utilizes the crosshair to identify a display symbol, and presents information about the vector indexed.

When indexing a cluster plot, the user moves the crosshair to the center of the desired display symbol and enters a "c". In the case of a scatter plot, the crosshair is used twice, first to select a lower left-hand corner, and then to select an upper right-hand corner of a "box". The desired information is printed for all the projected vectors in this "box".

For a histogram, the user moves the crosshair to a specific bin and enters a "c" as before; information is obtained for all vectors in that bin.

Function Description: index can be called with parameter "count" or "id". For a cluster plot the default value is "count ", while for the scatter plot, the only value accepted is "id ", which is the default value. A parameter is expected when indexing a histogram; however, the only value accepted is "count".

"count" causes the number of vectors in each class for a specified grid to be printed, while "id" produces each individual vector ID number.

The "box", which is constructed for a scatter plot, can be as large as desired but must enclose at least one complete class symbol.

The output for a one-space plot is the count or probability of each class present in the specified bin.

Function Call: intensfy (classlist)

Input parameters:

classlist	This is an optional list of class symbols, separated by blanks, representing classes to be intensified.

Additional User Interaction: None

Function Description: intensfy highlights a class or classes currently displayed by drawing a solid outline around the given class distributions. If there are no parameters input by the user, then all classes of the current data set will be intensified. It should be noted that, unless classes are well separated, if more than two classes are concurrently intensified, then the display itself becomes cluttered with lines and it is hard to observe the distributions. This routine is applicable to one-space displays only.

Function Call: latclogc (treename)   (classname)

Additional User Interaction:

** The logic tree associated with the selected data set is displayed.

Input the number of logic nodes to be connected.
3

Enter 3 logic node numbers on 3 separate lines.
5
11
2

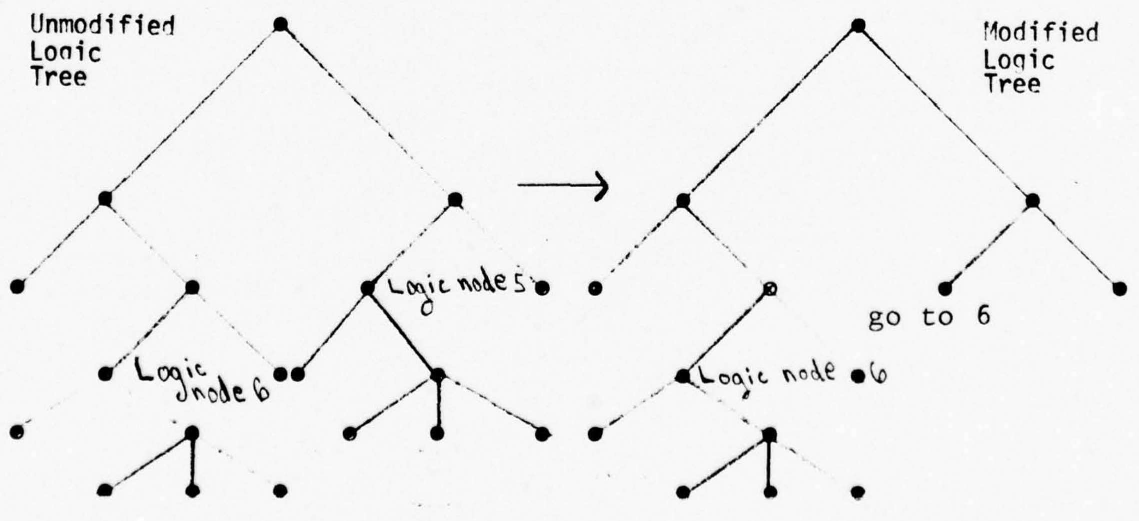
logic nodes 5 and 11 have been linked to logic node 2.

Function Description:

latclogc allows the user to modify a selected logic tree such that more than one path may be taken to arrive at a given logic node. Logic trees may be created with a "lattice" type structure through use of this option.

Any nodes linked through latclogc must have the same classes present. Also, no logic node may be connected to a logic node superior to it in the logic tree.

In the diagram below, the substructures below logic nodes 5 and 6 are identical and were linked by latclogc.



It should be noted that the order of logic creation can be important when using latclogc. Consider the following example: The user has constructed a logic tree with three logic nodes which he wants to link via latclogc. If no logic has been designed at any of these logic nodes, latclogc will cause all the vectors associated with the three logic nodes to be associated with the logic node whose node number is the smallest. Further logic designed at this node will include all the vectors from the three logic nodes.

If further logic has already been designed at one of the logic nodes, however, the remaining two will be linked to this logic node. The vectors associated with the remaining two logic nodes will not be used in any further design of logic.

Function Call: linglogc (treename) (classname)

Additional User Interaction:

```
enter Boolean statement for partition logic
Boolean statement
if the above statement is true, what classes should be
  assigned?
ab...f  (display characters)
the logic node assigned to these classes is n
if false, what classes should be assigned?
xy...z  (display characters)
the logic node assigned to these classes is m

**Compilation follows:

     (error messages from compilation of Boolean statement,
      if any)
was compilation successful?
yes/no
```

The standard group logic partial evaluation output is then displayed on the screen, with an option to hardcopy it on the printer along with a listing of misclassified vectors.

Function Description: linglogc creates Boolean or linguistic logic at a specified logic node.

The Boolean statement takes the form of a logical and/or arithmetic expression about the measurements in the selected data set. Measurement i is referred to as m(i). The statement may not exceed 132 characters and must be on one line. Examples:

a) $m(1) < m(2)$

b) $m(1)/m(10) = \cos(m(4))$

If any error messages appear while the entered Boolean statement is being compiled, the compilation was unsuccessful. In this case linglogc should be invoked again and the Boolean statement correctly reentered.

Function Call: lingpart (treename) (classname)

Additional User Interaction:

```
Enter Boolean statement
Boolean statement
if the above statement is true, what should be new node name
4-character node name
if false, what should be new node name
4-character node name
PL/1 version 2
was compilation successful and/or do you wish to continue?
yes/no
```

The numbers of vectors assigned to the true and the false side of the statement are given.

Function Description: lingpart divides a lowest data class into 2 subclasses based on criteria given in a Boolean or linguistic statement.

The Boolean statement takes the form of a logical and/or arithmetic expression about the measurements in the selected data set. Measurement i is referred to by the notation m(i). The statement may not exceed 132 characters and must be on one line. Examples:

a) m(3) > = 2*m(1) + m(2)
b) (m(1) + m(2)) < (m(3) + m(4))
c) m(1) = sqrt(m(2)) ("=" being logical and not arithmetic)

If any error messages appear while the entered Boolean statement is being compiled, compilation was unsuccessful; in such a case, when the question relating to compilation is asked, "no" should be entered.

Function Call: lingrjct (treename) (classname)

Additional User Interaction:

**Upon initiation, the program displays the current logic tree.

Input number of nodes where reject strategy is to be implemented, followed by logic node numbers - one to a line.

number of nodes
first logic node number
.
.
.
last logic node number

Is the reject strategy to be the same for all nodes?
yes/no

Enter Boolean statement for reject logic at node n
Boolean statement

**At this point an error message (if any) for the entered statement will be produced.

Is this correct?
yes/no

Function Description: lingrjct appends an independent Boolean reject strategy to any node in a MOOS logic tree. The independent reject test is made prior to the execution of logic at a node, i.e. it is done before the "normal" logic is performed at the node.

The Boolean statement takes the form of a logical and/or arithmetic expression about the measurements in the selected data set. Measurement i is referred to as m(i). The statement may not exceed 132 characters and must be on one line. Examples:

a) m(2) > m(3) + m(4)

b) sin (m(12) * m(6)) < = .67

The program permits the addition of the same reject logic to several nodes simultaneously, or of different independent reject strategies to each selected logic node.

Independent reject strategies which prove to be unsuccessful may be removed by using <u>deletlog</u>.

<u>NOTE:</u> <u>A vector will be rejected if it fulfills the conditions of the Boolean statement.</u>

Function Call: list_cst

Additional User Interaction: None

Function Description: This program lists all data trees that are currently in the common-access "trandata" directory.

Function Call: list_ust

Additional User Interaction: None

Function Description: This program lists all data trees that are currently in the user's "saved_trees" directory.

<u>Function Call</u>: <u>listlogc</u> (treename) (classname)

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>listlogc</u> produces a printout of a user-specified logic tree on the high-speed printer. It outputs all tests and branches contained in the logic defined by the user for treename-classname in an easy-to-read format, listing the values of all logic parameters and the possible paths that may be followed in logic evaluation.

Function Call: log$dlet (treename) (classname)

Additional User Interaction: None

Function Description: log$dlet allows the user to delete any saved logic file from permanent storage. Error messages will be printed if no saved logic file exists in permanent storage for the input reference pair, or if the file is not deleted.

<u>Function Call</u>: <u>log$list</u>

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>log$list</u> allows the user to determine the number of saved logic files he has in permanent storage, along with their reference name pairs.

Function Call: log$rstr (treename) (classname)

Additional User Interaction: The user will be asked to supply a new treename-classname pair if a logic file already exists in his process directory under the input reference.

Function Description: log$rstr allows the user to restore any saved logic file from permanent storage to his process directory. The saved logic file still exists in permanent storage at the completion of this procedure. Error messages will be printed if no logic file exists in permanent storage for the input reference pair, or if copying is unsuccessful.

Function Call: log$save (treename) (classname)

Additional User Interaction: The user will be asked to supply a new treename-classname pair if a saved logic file already exists under the input pair reference.

Function Description: log$save allows the user to save any logic file in his process directory by copying it into permanent storage. Error messages will be printed if no logic file exists for the input reference pair or if copying is unsuccessful.

Function Call: logicevl (treename) (classname)

Additional User Interaction:

Input the name of the data set on which logic was designed
treename classname

**At this point the confusion matrix display as described in Section 1.3.3 is presented

Do you want a hardcopy of this matrix?
yes

Do you want a listing of incorrectly classified vectors?
yes

Function Description: The routine logicevl enables the user to test completed logic. Any data set may be evaluated against logic designed on any other data set by using this function (assuming both data sets have the same dimensionality).

Individual vectors may be tested against completed logic by setting sense switch 4 prior to running logicevl. All vectors to be evaluated must then be specified by vector I.D. number.

In the event that the class names of the test set are not the same as the class names of the design set, the function reasname may be invoked before proceeding with logic evaluation. For a detailed discussion of the reassociated name capability, see Section 1.3.3.

Function Call: measxfrm (treename) (classname)

Additional User Interaction:

```
enter a new treename
8-character treename
enter dimensionality of the new tree
number
enter transformation expressions
up to 75 transformation character expressions
.q(to terminate entering expressions)
PL/1
was compilation successful and/or do you wish to continue?
yes/no
```

Function Description: measxfrm is a means of transforming one data set with dimensionality m into another data set of dimensionality n (n may or may not = m). This transformation is done by means of character arithmetic expressions. Measurement i in the new tree is symbolized by nm(i). Measurement i in the old tree is symbolized by om(i).

For example, suppose we have a tree named tree0001 with dimensionality four and wish to create a tree named tree0002 with dimensionality five. Furthermore, suppose each measurement in tree0002 was to be the same as tree0001 with the exception that measurement five of tree0002 was to equal the sum of measurements three and four of tree0001. Interaction would go as follows:

```
measxfrm  tree0001
enter a new treename
tree0002
enter dimensionality of the new tree
5
enter transformation expressions
nm(1) = om(1)
nm(2) = om(2)
nm(3) = om(3)
nm(4) = om(4)
nm(5) = om(3) + om(4)
.q
PL/1
was compilation successful and/or do you wish to continue?
yes
```

The ".q" signifies the end of entering transformation expressions.

This routine makes use of the MULTICS PL/1 compiler. Therefore, the more the user knows about PL/1, the easier will be the job of constructing transformations. For example, the above transformation could have equivalently been written

```
do  i = 1 to 4
nm(i) = om(i)
end
nm(5) = om(3) + om(4)
```

The following initial conditions are always established by this routine and may be of help to the user. Suppose the data set under transformation has dimensionality m and the new data set has dimensionality n.

```
let  j = min(m,n)
set nm(i) = om(i) for all i ≤ j

if n > m
set nm(i) = o  (m+1 ≤ i ≤ n)
```

However, statements entered by the user override any of the above initial conditions.

Function Call: mergmeas treename

Additional User Interaction:

```
type in names of two trees to be used in making new tree.
one per line
oldtree1
oldtree2
do you wish to check ID's to make sure vectors are in
the same order in each node of the two data trees?
yes
```

Further dialogue is needed in the case where vectors are slightly out of order. For example, a vector may be missing in one node in one tree and the user may wish to delete it from the corresponding node in the other tree.

Function Description: The measurements of the vectors in the second tree are concatenated to the measurements of the corresponding vectors in the first tree, producing a new tree (treename) whose structure is identical to the two original trees, but whose dimensionality is equal to the sum of the dimensionalities of the two original trees.

Possible Errors: mergmeas will exit if the number and names of the lowest nodes in the two original trees are not the same, if the numbers of vectors in the corresponding nodes of the two original trees are not equal, if the dimensionality of the new tree is greater than 100, or if either of the two original trees is not known to the system. In these cases the routine will return with the current data set "notatree", "nono".

Function Call: moosmode (treename) (classname)

Additional User Interaction:

```
There are currently two tree(s) whose dimensionality is
  100.

     tree0001   120
     tree0002   125

These trees must be deleted from the system or their
dimensionality reduced to 100 or less before regular
MOOS operation may begin.

There are currently three tree(s) whose mean and covariance
values may be calculated at this time.

     1   tree0003   20
     2   tree0004   45
     3   tree0005   10

Any or all of these trees may be converted to normal
MOOS trees.  Enter number of trees to be converted, or
"all."

all
```

Function Description:

The chief function of moosmode is to calculate the mean vectors and covariance matrices for any trees for which these quantities have not been calculated, i.e., trees created through use of the excess measurement mode. The majority of MOOS functions may not be executed until this is done (see Section 1).

The calculation of mean vectors and covariance matrices is not allowed on any trees whose dimensionality is greater than 100. The system will remain in the excess measurement mode until there are no trees whose dimensionality is greater than 100.

moosmode may also be used simply to list excess measurement mode trees currently existing in the system. The user would request, in this case, that zero trees be converted to normal MOOS trees.

Function Call: nlm (treename) (classname)

Additional User Interaction:

```
        enter total number of cluster centers
        number
        do you want an equal number of cluster centers in each
        class(e) or the number of cluster centers based on original
        data distribution(o)
        e/o

        **At this point, the clustering routine displays a table
        of information concerning cluster center radii

        enter new treename
        new treename

        Do you want to project onto a two-space or a three-space?
        2/3
        Do you want to project onto the coordinate plane with
        max variance?
        yes/no
        The data will be projected on the following dimensions:
        d1 d2 (d3)
        Error for iteration 1 is (value)
                  .
                  .
                  .
        Error for  iteration  10 is (value)
        Type "d" to see current mapping "m" for more iterations
        m

        How many more iterations?
        2
        Error for  iteration  11 is (value)
        Error for  iteration  12 is (value)
        Type "d" to see current mapping "m" for more iterations
        d
        **At this point, a scatter plot of the current mapping is
        displayed

        Type "a" to accept, "m" for more iterations
        a/m
```

Function Description: nlm (nonlinear mapping) is a structure analysis routine which maps data vectors from N-space to two-or three-space while attempting to preserve the N-space "structure" of the data. Due to the time and space-consuming nature of the algorithm, a limit of 200 vectors has been set. If the selected data set has more than 200 vectors, a data-clustering routine is automatically called which forms a new MOOS data tree whose fewer vectors are, hopefully, representative of the original data set. The <u>nlm</u> algorithm itself begins at this point on the "clustered" tree. An association between the clustered tree and the original data tree is maintained. This means that any restructuring done on the clustered tree is also carried out on the original data tree.

Throughout the execution of <u>nlm</u>, the user is given the choice of more iterations to make the mapping more accurate, or of viewing the current mapping. The relative error function should eventually become small and tend to "level off." A point should soon be reached when further iterations would cause little improvement in the mapping.

If the three-space option is chosen, the projection shown throughout the execution of <u>nlm</u> will be on the first two of the three coordinate axes. When the algorithm is complete, the other pairs of axes may be viewed by typing <u>seq</u>.

Function Call: nmv (treename) (classname)

Additional User Interaction:

Enter an option:

1 simple nearest mean vector
2 inverse variance weighting (weighting vector)
3 Mahalanobis (weighting matrix)
n
Do you wish to implement any reject boundaries?
no

Function Description: nmv generates nearest mean vector logic (see Section 1.3.3.2.1) based on various user-specified options. After partial logic evaluation, the user may choose to accept the results of the evaluation, or recreate the logic with a different set of options.

Options:

Option 1. The simple nearest mean vector option causes a vector to be assigned to the class whose mean the vector is closest to in Euclidean distance.

Option 2. The weighting vector option operates in the same manner as simple nearest mean vector except that each dimension is weighted by the inverse of the class variance along that dimension.

Option 3. Mahalanobis weighting is weighting the distance calculations by the inverse covariance matrices of the classes.

The user may set a reject distance for all classes, or a specific reject distance may be entered for each class.

Function Call: nmvmod (treename) (classname)

Additional User Interaction: The user is asked to select which nearest mean vector logic node is to be modified and then to enter options for that node.

Function Description: nmvmod is designed to cycle through all nearest mean vector logic nodes, retrieving nodes for modification, presenting evaluation options, and performing partial logic evaluation (if requested). The program will exit if no MOOSLOGIC file exists for the treename-classname pair, if there are no nearest mean vector logic nodes, or if the user enters a 0 in response to the option for selection of a nearest mean vector logic node. See program nmv for details on the options available.

Function Call: normxfrm (treename)

Additional User Interaction:

Input new treename
treename

Function Description: normxfrm produces a new data tree which is a normalized version of the selected data tree. The normalization procedure involves dividing each measurement of each vector by the standard deviation of that measurement for the entire data tree. The variance of each measurement in the new data tree is, therefore, one.

<u>Function Call</u>: <u>page</u>

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>page</u> allows the user to page through a rank-order display. Up to 50 measurements can appear on the screen at one time, and thus, if there are more measurements to be seen, the <u>page</u> option permits the user to see them.

Function Call: pairmod (treename) (classname)

Input Parameters: treename and classname are optional and designate a data set other than the current one.

Additional User Interaction: If there is an error associated with the input data, one of the following self-explanatory messages is printed and the user is returned to the command level:

1) "no logic file currently exists for this data set"
2) "dimensionality and/or number of low nodes of data files and logic files are not equal"
3) "no completed pairwise nodes exist"
4) "current logic node is illegal"

The user is first asked whether there are any logic nodes to be combined. If the user responds "yes", the Fisher logic is recomputed with some of the data classes combined. Further logic may then be designed to separate the classes that were combined.

The classes at the user-specified Fisher node are then listed and the user is asked to input the pair to be modified.

"Enter class pair to be modified (on one line-no delimiters)"

For example, to adjust the logic for the pair nod1 versus nod2, the correct response is "12."

If this information is entered incorrectly, the following is printed:

"a/b is an invalid class pair;
Do you wish to continue?"

If the answer to the question is "yes," execution of pairmod recommences. A "no" response brings the user back to the command level.

Based upon the existing logic for the selected class pair, a list of options is presented; the user should enter the number of the desired option.

OPTIONS

A) "Change the number of thresholds"

This option allows a user to change the number of Fisher thresholds used in evaluation of a specific pair or of all pairs of a Fisher node.

The user is asked

"Is this adjustment to apply to all pairs?" A "no" response will change the number of thresholds for the selected pair. The user then enters the new number of thresholds.

"The current number of thresholds used is n"
"Enter the number of thresholds"

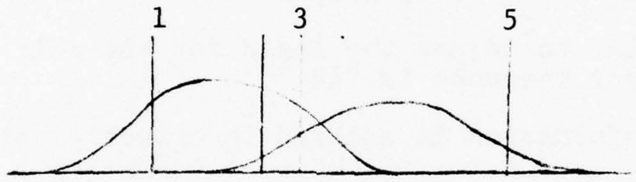
If this new number is not valid, the following is printed:

"Number of thresholds must be between one and four"
"Enter the number of thresholds"

The user should then reenter the number of thresholds.

B) "Change the location of thresholds"

This option is valid if the present logic for this pair is Fisher or arbitrary one-space. The data is projected upon the appropriate basis vector and the threshold(s), either one or two for arbitrary one-space, or five for Fisher, are represented by vertical lines. If the logic is Fisher and the threshold is being used in evaluation, a number representing its sequential position from left to right on the screen is printed above the vertical line. For example, if the current logic was Fisher with three thresholds being implemented, the display would be:



These numbers do not appear above existing arbitrary one-space thresholds.

A numbered list of display options will appear in the upper right-hand corner of the screen. The list of options below follows with a brief description of each, a letter that indicates which type of logic is applicable (F for Fisher, 0-S for arbitrary one-space), and which user program contains more detailed information about the operation of each option.

(F,0-S)	1.	"select cr" see select	change the range of the display
(F,0-S)	2.	"select cb" see select	change the number of bins
(0-S)	3.	"dra$bndy" see dra$bndy	draw a threshold
(0-S)	4.	"dboundry" see dboundry	delete a threshold
(F,0-S)	5.	"display A" see select	display class A of pair A/B

(F,O-S) 6.	"display B" see select	display class B of pair A/B
(F,O-S) 7.	"display A,B" see select	display both classes.
(F,O-S) 8.	"index count" see index	statistical information for a particular bin.
(F,O-S) 9.	"hgprint" see hgprint	copy display to printer.
(F,O-S)10.	"intensfy" see intensfy	intensify both classes.
(F)11.	"move thres"	move threshold(s). This option must be selected to move any thresholds for Fisher.
(F,O-S)12.	"continue"	resume execution of pairmod. This option must be the last selected. If the user does not want to modify the first display, this option number must be entered to continue with pairmod.

The user enters the desired option number.

To draw thresholds when the logic is arbitrary one-space, option number 3 is used. When the user has finished drawing thresholds, the "continue" option number (i.e. 12) should be entered.

When the Fisher logic exists and the display has been examined and modified as desired, the user should enter the "move thres" option number (11). The following dialogue then occurs.

"Enter number of thresholds to be moved" n
"Send character corresponding to the number of the threshold to be moved"

The cursor will be activated n times. Each time, the user moves it to the new threshold position and enters the appropriate number, from the numbers 1 thru 5 above the boundaries, that corresponds to the threshold to be adjusted.

For arbitrary one-space logic the user is then asked

for 1 boundary:

"Enter class present on right of boundary"
"Enter class present on left of boundary"

for 2 boundaries:

"Enter class present on right of right boundary"
"Enter class present on left of left boundary"
"Enter class in middle region"

The user should send the correct class symbol, or "****" to designate a reject region.

C) "Change the number of measurements"

This option allows a user to modify the number of features used to determine the Fisher direction. With this option, there is the capability of also modifying the number of thresholds to be used in evaluation. The user is asked:

"Enter the number of thresholds to be used in evaluation"

To retain the present number used, the user must input this value.

D) "Change the number of and/or location of boundary(ies)"

The two-space plot is generated and the user is presented with the following list of display options. The list below contains brief descriptions of each option and tells which user program describes the operation of the option in more detail.

1.	"scale$zm" see scale$zm	enlarge a subarea of display
2.	"scale$rt" see scale$rt	return to "original" data ranges
3.	"dra$bndy" see dra$bndy	draw boundaries
4.	"dboundry" see dboundry	delete boundaries
5.	"display A" see elimclas	display class A of the pair A/B
6.	"display B" see elimclas	display class B of the pair A/B
7.	"display A,B" see elimclas	display both classes
8.	"index count" see index	obtain the number of vectors in a cluster plot grid
9.	"index id" see index	obtain vector ID numbers
10.	"cdisplay" see cdisplay	change the display from cluster to scatter and vice versa
11.	"clprint" see clprint	copy the "cluster" plot to the printer
12.	"continue"	resume execution of pairmod.

Before drawing new boundaries, the user must delete the existing ones by entering option number 4. When the display has been modified and the new boundaries drawn, the following dialogue occurs:

<u>for one boundary</u>:

"Enter the class present on the convex side of boundary 1"
"Enter the class in excess region"

<u>for two boundaries</u>:

"Enter the class present on the convex side of boundary 1"
"Enter the class present on the convex side of boundary 2"
"Enter the class in excess region"

The appropriate class symbols should be entered, or "****" if a reject region is desired.

E) "Change to Fisher"

This option is only applicable after the logic has been changed from Fisher to some other type.

The type of Fisher logic returned is the latest version. For example, if the logic sequence had been: a) alter the number of measurements for determining the Fisher direction, and b) optimal discriminant plane, the Fisher direction used by this option is the direction calculated when <u>fisher</u> was last called. To return to the "original" Fisher direction, the MOOS function <u>fisher</u> has to be called again.

The user is requested to:

"Enter the number of thresholds to be implemented." This is the number of thresholds used in evaluation.

F) "Change to arbitrary one-space"

The logic enables the user to employ a previously saved vector as a projection vector. The user is asked to enter a vector or supply the name of a vector which has been stored via the utility function <u>vec$save</u>.

The data is then projected upon the basis vector and the histogram appears with the display option list as described under option B. By entering appropriate option numbers, the user can manipulate the display and construct one or **two** thresholds. When he is finished, the option number for "continue" should be entered. The user is then asked:

for one boundary:

"Enter class present on right of boundary"
"Enter class present on left of boundary"

for two boundaries:

"Enter class present on right of right boundary"
"Enter class present on left of left boundary"
"Enter class present in middle region"

The user should reply with the class symbols, or "****" to create a reject region.

G) "Change to optimal discriminant plane"

The discriminant plot is generated with the display option list equal to that under logic option D. Upon completion of display manipulation and boundary construction, the user will proceed to answer the dialogue as described under option D.

H) "Change to arbitrary two-space"

The user supplies either the two basis vectors or the names of vectors entered by the MOOS utility function vec$save.

The user then proceeds in the manner described under logic option D.

I) "Change to/modify Boolean"

The user can supply or modify an existing Boolean statement for use in pairwise logic evaluation. The maximum length for the statement is 132 characters. The user is asked:

"Enter Boolean statement for partition logic"

After the statement is entered the user is asked:

"If above statement is true, what class should be assigned?"

The user's response to this is the class symbol of the class desired.

Upon completion of a specific logic option, a "mini" confusion matrix, consisting only of the pair involved, is printed, and the user is asked:

"Is this logic acceptable for the pair?"

No existing logic is modified until the answer to this question is "yes".

Then the user is asked:

"Is there another pair to be modified?"

A "no" response produces the confusion matrix of the entire Fisher node, while a "yes" response makes pairmod recommence execution.

Function Description: Through pairmod, the logic for any or all pairs of classes of a pairwise logic node can be altered. The user can make as many changes as desired (for example, using the Fisher logic with the four different numbers of thresholds, moving thresholds, examining the discriminant plane, and any arbitrary two-space) with or without having the current pairwise logic modified.

pairmod saves the current version of Fisher logic and the current modified logic, but only those two. In the above sequence, the Fisher and the arbitrary two-space logic would be saved while the intermediate designs would be destroyed.

AD-A034 393 PATTERN ANALYSIS AND RECOGNITION CORP ROME N Y F/G 9/2
MULTICS OLPARS OPERATING SYSTEM. VOLUME I.(U)
SEP 76 D B CONNELL, K N KLINGBAIL F30602-75-C-0226
UNCLASSIFIED PAR-74-25-A RADC-TR-76-271-VOL-1 NL

3 OF 3
AD
A034393

END
DATE
FILMED
2-77

Function Call: probconf (treename) (classname)

Additional User Interaction:

```
do you want the default interval range of 3 standard
deviations?
no

should the interval range be calculated using a number
of standard deviations (s), or the absolute range of
the data (r)
type in s or r
r
**At this point, a table is presented which contains the
  number of bins, interval size, lower bound, upper bound,
  and range for each measurement
Do you want to change the interval size?
no

**At this point, a rank order display is presented
```

Function Description: probconf produces a measurement evaluation for the selected data set based on a histogram estimation of the marginal class conditional probabilities. The values produced are measures of the overlap of these probabilities; therefore, the smaller the value, the better the measurement.

Probability histograms used in this calculation may be viewed directly by selecting the histgram option.

The dialogue is designed to allow the user to select the interval range and number of histogram bins which will best represent the data distribution.

Function Call: rdisplay

Additional User Interaction: None

Program Description: rdisplay reconstructs the most recent two-space plot, one-space plot, or confusion matrix (through a call to displacm).

Function Call: reasname (treename) (classname)

Additional User Interaction:

**Upon initiation, a table is printed which contains each lowest logic node number, the class name which belongs at that node by design, and a "reassociated" name.

Enter number of logic nodes whose reassociated names are to be changed
n
Enter n logic node numbers and corresponding new reassociated names.
2 name(1)
3 name(2)
.
.
.
m name(n)

**At this point, the original table is redisplayed with the input changes

Are these reassociated names correct?
yes/no

Function Description: reasname allows the user to change reassociated names to whatever he desires. This is useful in cases where a test data set is to be evaluated against logic designed on a tree which had different class names. A more complete description of the purpose of reassociated names can be found in Section 1.3.3.

Function Call: redraw

Additional User Interaction: None

Function Description: If a boundary has been drawn on a one-or two-space plot and a new plot of the same data put on the screen, the boundary does not automatically reappear. redraw reconstructs the boundary, and in the case of two-space, it extends line segments to the edge of the projection.

redraw also reconstructs boundaries drawn on the original projection onto "zoomed" projections, and vice versa.

Function Call:	remtree (treename/"all")

Input Parameters:

treename	specify a particular data set
"all"	delete all existing trees from the "trandata" directory

Function Description: This routine deletes any or all trees and nodes under these trees that have been saved, via the utility function savec, in the common-access "trandata" directory.

The user is cautioned against the use of the "all" parameter, as all trees, regardless of which user saved them, will be removed.

Function Call: **restore** treename/"all"

Input Parameters:

treename	specify a particular data set
"all"	copy all existing trees in the user "saved_trees" directory

Additional User Interaction: If the name of any data set that is being copied is already in the process directory, an error message is printed and the user is asked to enter a unique eight-character treename. The data set will be restored under that name.

Function Description: This routine returns from the user "saved_trees" directory any or all data sets that have previously been saved via the utility function save.

<u>Function Call</u>:	<u>restorec</u> treename/"all"

<u>Input Parameters</u>:

treename	specify a particular data set
"all"	copy all existing trees in the "trandata" directory

<u>Additional User Interaction</u>: If the name of any data set that is being copied is already in the process directory, an error message is printed and the user is asked to enter a unique eight-character treename. The data set will be restored under that name.

<u>Function Description</u>: This routine returns any or all data sets from the common-access directory "trandata" to the process directory.

These trees must have been stored via the utility function <u>savec</u>.

Function Call: restruct (treename)

Additional User Interaction:

> type in the name of the lowest node to be restructured
> from the following list
> node(1) node(2) ... node(n)
> nodename
>
> input 3 new 4-character node names
> newname(1)
> newname(2)
> newname(3)

Function Description: After a boundary has been drawn on a one-space or two-space plot, the user must invoke restruct so that data vectors will be relabeled according to the boundary(ies).

The two-space naming convention is as follows: the first name input is the name of the class on the convex side of the first boundary drawn. If there are two boundaries, the second name input is the name of the class on the convex side of the second boundary drawn. The last name input is always the name of whatever region remains.

In one-space, input names will refer to the boundary-separated regions from left to right.

Function Call: rnk$bcls classname or unique class character

Additional User Interaction: None

Function Description: rnk$bcls ranks the measurements associated with the current data set in order of their effectiveness in discriminating the selected class from all other classes (as determined by dscrmeas or probconf). See Section 1.3.1.

Function Call: rnk$bycp class1 class2

Input parameters:

class1 and class2 are class names or unique class characters

Additional User Interaction: None

Function Description: rnk$bycp ranks the measurements associated with the current data set in order of their effectiveness in discriminating "class1" from "class2". See Section 1.3.1.

Function Call: rnk$mbc measurement number

Additional User Interaction: None

Function Description: rnk$mbc ranks the classes in the current data set according to the effectiveness of the selected measurement in discriminating each class from all others (as determined by dscrmeas or probconf). See Section 1.3.1.

Function Call: rnk$mbcp measurement number

Additional User Interaction: None

Function Description: rnk$mbcp ranks the possible class pairs in the current data set according to the effectiveness of the selected measurement in discriminating between the classes in each pair (as determined by dscrmeas or probconf). See Section 1.3.1.

Function Call: rnk$oall

Additional User Interaction: None

Function Description: rnk$oall gives an overall ranking by means of running dscrmeas or probconf. An ordered list of the measurements is given (a result of the overall ranking calculations) along with the class and class pair best discriminated (or least confused) for each measurement. Refer to Sections 1.3.1.1 and 1.3.1.2 of this report for a mathematical discussion.

Function Call: save treename/"all"

Input Parameter:

treename	specify a particular data set
"all"	copy all existing trees in the process directory

Additional User Interaction: If the name of any tree that is saved already exists in the user's "saved_trees" directory, an error message is printed and the user is asked to enter a unique eight-character tree name. The data set will be saved under that name.

Function Description: This program copies any or all trees in the "sysdata" file into the user's "saved_trees" directory. This directory is only accessible by the particular user who created it.

For other users to be able to access this data, it must be stored in the "trandata" directory via the savec utility function.

Function Call: savec treename/"all"

Input Parameters:

treename	specify a particular data set
"all"	copy all existing trees in the process directory

Additional User Interaction: If the name of any data set that is being copied already exists in the common-access "trandata" directory, an error message is printed and the user is asked to enter a unique eight-character tree name. The data set will be saved under this name.

Function Description: This program copies any or all trees in the "sysdata" file into the "trandata" directory, where other users can access these data sets.

To save these data for the specific user's exclusive reference, the utility function save should be used.

Function Call: scale$zm

Additional User Interaction: Using the crosshair, the user must select the portion of the current display to be "zoomed" upon. When the crosshair is on for the first time, the user positions it at the lower left-hand corner of the new display area and enters a "c ". The second time, the user positions it at the upper right-hand corner of the new display area, and enters another "c ".

Function Description: scale$zm allows the user to select a subarea of the current display and have a closer, more detailed examination of that area. In the manner described above, the user first selects the lower left-hand, then the upper right-hand, corner of the area to be magnified. The new display is that subarea.

Function Call: scale$rt

Additional User Interaction: None

Function Description: scale$rt is the complement of scale$zm. This routine allows the user, upon completion of one or several zooming operations, to return to the display with the original x-and y-ranges.

<u>Function Call</u>: <u>sel$meas</u> $meas_1$ $meas_2$ ... $meas_n$

<u>Additional User Interaction</u>: None

<u>Function Description</u>: <u>sel$meas</u> enables the user to select a given set of measurements. An "*" will appear next to all selected measurements. However, if an "*" already exists for a measurement, selecting that measurement again will cause the "*" to be turned off. This routine is used in conjunction with <u>rnk</u> routines and the routine <u>trnsform</u> to transform a data set. All "*"ed measurements are used in the transformation.

Function Call: sel$thrs value

Additional User Interaction: None

Function Description: This routine is similar to sel$meas, except that a threshold value is entered; all measurements whose value is greater (for dscrmeas) or less (for probconf) than the entered value will appear with an "*" next to them.

Function Call: select (macro/micro) (classlist) (cr) (cbnnn) (prob/count)

Input Parameters:

macro/micro	The display format will be changed according to the user's selection.
classlist	The class list is a set of data class symbols separated by commas or blanks. The classes seen in subsequent displays will be only those listed.
cr	The cursor will be activated to allow selection of range points representing the minimum and maximum data desired in the display. The original scale will be restored when the second range point selected is to the left of the first range point. Selected range points will be indicated by moving the vertical crosshair to the desired location and entering a "c ".
cbnnn	The cb option resets the number of bins desired in the data presentation to nnn, e.g. cb50 results in a 50-bin display.
prob/count	The area under each class histogram is affected by the selection of prob or count in the parameter list. Under the count option, each histogram column is proportional in area to the number of vectors in each displayed class which falls into a given bin; therefore, the total area under an entire class histogram is indicative of the number of vectors in each class. The prob option produces histogram columns representing the percentage of each class within a given bin; therefore, the total areas under all class histograms are equal to one another (that is, totals to 100 percent).

Additional User Interaction: None

Function Description: select is the major one-space display utility routine. The one-space algorithms will default to a macro display of all classes, if there are more than three classes in the data tree, or to a micro display if there are three or less classes in the data tree. If there are more than 18 classes in the data tree, the first 18 are displayed and the user is asked if he would like to see the remaining classes. A "yes" response erases the screen and presents the other classes; a "no" response terminates the routine. The type of display, either macro or micro, will remain the same until the other is specified. When the alternative is specified, all classes are displayed unless a class list is also input. In that instance, only those classes specified are displayed. The class list consists of the display characters separated by commas or blanks. All parameters are also to be separated by blanks.

Parameter cr indicates "change range ". If this option is selected, the crosshair is turned on. The user then moves it to the desired new xmin and sends a "c ". The crosshair then returns for user selection of a new xmax and again accepts a "c ". If, having already zoomed on a display, the user wishes to return to his original display, cr is again the appropriate parameter. When the crosshair is turned on, the new xmax is placed to the left of the new xmin ($xmax < xmin$). This will cause regeneration of the original range.

The cbnnn parameter changes the number of bins, where nnn is the new integer number of bins. There is no blank between "cb" and "nnn". The original display will be recreated only via another cbnnn call, where nnn is the original number of bins.

The prob/count parameter is applicable only to the micro view. The selection of this option changes the scaling in the micro view either to probabilities or to counts of each bin. The macro view uses probabilities where the largest "spike" or value corresponds to the maximum probability of a bin for all classes that are currently displayed. Counts are default in the micro view.

The following is an example of how to manipulate the display, given that there are five classes A,B,C,D and E:

Initial display: (from crdv$sal, for example) macro view, all classes displayed, range is the overall range of data along some coordinate, N bins and probabilities scaling.

select c cr	macro view, crosshair turned on, class C is the only class displayed, probabilities scaling

select macro	macro view, all classes displayed, "current" range, probabilities scaling
select micro D	micro view, class D only displayed, "current" range, count scaling
select micro B C cb75 prob	micro view, classes B and C displayed, "current" range, 75 bins instead of original N, y-axis probabilities scaling
select macro cr cbN	crosshair turned on, return the number of bins to N, macro view. This can return to the original display. In a macro view, probabilities scaling is always used; thus, the display scaling is probabilities, although it was not specified with the prob parameter.

Function Call: a) sense number on
b) sense number off

Additional User Interaction: None

Function Description: Function sense provides user control over the sense switch settings. Sense switches may be used to control any dynamic options as required by the system designer. The sense switch numbers range from 1 to 36.

Currently assigned switches:

1 - If this switch is set, overall logic evaluation (logicevl) will produce a listing of pairwise vote counts for all vectors which are assigned to pairwise logic nodes, whether correct or not.

2 - If this switch is set, the test for correctness of a vector in overall logic evaluation is performed on the display symbols of the classes involved, rather than on the full four-character names.

3 - Used as an internal flag to indicate the excess measurement mode.

4 - If this switch is set prior to running logicevl, all vectors to be evaluated must be specified by vector I.D. number.

5 - Used as an internal flag by dg$dcrmsu and dg$dd.

6 - Used as an internal flag by features.

7 - Used as an internal flag by features.

8 - Used as an internal flag by fortlogc.

15 - Used by tapinput as an internal flag.

Function Call: seq

Additional User Interaction: None

Function Description: seq may be used after the following MOOS functions:

a) eigv$sa1, eigv$sa2

The ordered list of eigenvalues reappears and the data will subsequently be projected on a newly selected eigenvector(s).

b) nlm (three-space)

seq is used to sequence through the three possible pairs of projection axes.

c) crdv$sa1, crdv$sa2

The user will be allowed to select a new coordinate axis (or axes) for data projection.

d) gndv$sa1, gndv$sa2

The ordered list of eigenvalues reappears and the data will subsequently be projected on a newly selected generalized discriminant vector(s).

Function Call: setdata filename

Additional User Interaction:

```
    Enter range values (first & last index)  number number
    Enter type of entry
    'char' - character string of length four
    'flot' - floating
    'intg' - integer
    'bits' - bit stream
    four-character option

        .
        .  (enter each word)
        .

    Request complete.  Do you want to read more?  (yes/quit)
                                                    quit
```

Function Description: setdata allows the user to insert data into any MULTICS file available to him in his process directory. The data may be inserted in any of four forms (character, floating point, integer, or bit stream).

Function Call: summrycm

Additional User Interaction: The user is asked if he desires a hardcopy of the information.

Function Description: summrycm outputs confusion matrix information to the screen in a summary format, listing percentages for the number of vectors correct, in error and rejected, by class and overall.

Function Call: tapeoput (treename)

Additional User Interaction:

```
     Input tape label
     label

     Input file number to be written
     1

     Tape (label) will be mounted on drive 1 with a write ring
     Tape ready

     Tree (treename) written into file 1
```

Function Description: tapeoput writes a MOOS data tree onto a seven-track, 556 BPI magnetic tape. The tree may be reentered to the MOOS system through a call to the tapinput function. The exact format of the data is described in the writeup on tapinput.

Function Call: tapinput treename

Additional User Interaction:

Input tape label: label
Input tape file number: number
Input data dimensionality for tree treename: number
[tape is mounted and selected file is read]

Do you want to input another tree from tape X?
no

Function Description: The tapinput function transforms a data set on a MOOS data tape into a system data tree. The multiple-file MOOS data tape contains one data set per file, each set composed of data vectors in the following 36-bit MULTICS format:

```
            ⎧ Measurement 1        (MULTICS floating point)
            ⎪ Measurement 2
            ⎪       .
vector 1   ⎨       .
            ⎪       .
            ⎪ Measurement ndim
            ⎪ Vector Index         (integer value)
            ⎩ Data class name      (four ASCII characters)
                    .
                    .
                    .
                    .

vector n   {
```

The physical records on the MOOS tape may be up to 1632 36-bit words in length. Each record must contain an integral number of vectors ≥ 1. The tape must be low-density (556 BPI), with no labels, header records, or any other excess words.

Function Call: treedraw (treename) (classname)

Additional User Interaction: None

Function Description: treedraw displays a selected MOOS data tree, showing the structural relationship between various intermediate and lowest nodes of the tree. If the selected class name is ****, the entire tree is displayed. Portions of the tree may be viewed by selecting other class names.

If any level of a tree contains more than 32 nodes, that level and any structure below it will not be displayed. Instead, lines indicating where this excess structure is located will be drawn.

If the first level below the senior node has more than 32 nodes, a message is printed and nothing will be drawn. The dataprnt "tree structure" option may be used to display a tree in outline form in cases where a tree is too large for treedraw.

Function Call: treelist

Additional User Interaction: None

Function Description: This program lists all data trees that are currently active in user's "sysdata" file in a given process.

Function Call: trnsform

Additional User Interaction:

    enter a new tree name
    eight-character tree name

Function Description: trnsform performs a transformation on the data tree that was most recently used for doing the measurement evaluation. The tree name of the new tree is the entered tree name (see Additional User Interaction). trnsform saves those measurements which have an "*" next to them, the "*" appearing as the result of using a measurement selection routine (e.g. sel$meas, sel$thrs, or un$bbc). The tree structure is copied from the old tree to the new tree.

Function Call: un$bbc

Additional User Interaction: None

Function Description: un$bbc works similarly to the sel routines in that it gives the user the ability to select certain measurements for transformation. un$bbc places an "*" next to those measurements that best classify or discriminate each class in the data set.

Function Call: un$bbcp

Additional User Interaction: None

Function Description: un$bbcp works similarly to un$bbc, except that un$bbcp places an "*" next to those measurements that best classify or discriminate each class pair in the data set.

Function Call: vec$del saved vector name

Additional User Interaction: None

Function Description: vec$del deletes from the saved-projection-vectors file the saved vector whose name is entered.

Function Call: vec$hall

Additional User Interaction: None

Function Description: vec$hall is a means for hardcopying to the line printer a complete listing of all saved vectors, including each vector's name, length, and components.

Function Call: vec$lall

Additional User Interaction: None

Function Description: vec$lall is a means for listing, on the display, the name and length of all saved projection vectors.

Function Call: vec$list saved-vector name

Additional User Interaction: None

Function Description: vec$list is a means for listing, on the display, information about one particular saved projection vector, including the vector's name, length, and components.

Function Call: vec$save

Additional User Interaction: Do you wish to save the x projection, the y projection, or both the x & the y projection? (x,y,xy)

> x
> enter a vector name
> a five- to eight -character name

Function Description: vec$save saves projection vectors for the user. For example, suppose that after a scatter plot is displayed, the user wishes to save the projection vectors; vec$save is the correct routine to use. The user can save just the x-projection, just the y-projection, or both the x- & the y-projection. Each saved projection vector has a unique name with up to eight characters (not to be confused with the eight-character tree name).